

# Universidad Autónoma de Madrid

## Escuela Politécnica Superior



## TRABAJO FIN DE MÁSTER

**SEGURIDAD DE APLICACIONES WEB BASADAS EN LAS  
TECNOLOGÍAS NODE.JS Y MONGODB: ESTUDIO Y CASO  
DE USO**

**Máster Universitario en Ingeniería Informática**

**Pedro Alberto Ruiz González  
Tutor: David Arroyo Guardado**

**Junio 2018**



# **SEGURIDAD DE APLICACIONES WEB BASADAS EN LAS TECNOLOGÍAS NODE.JS Y MONGODB: ESTUDIO Y CASO DE USO**

Autor: Pedro Alberto Ruiz González  
Tutor: David Arroyo Guardeno

Departamento de Ingeniería Informática  
Escuela Politécnica Superior  
Universidad Autónoma de Madrid

Junio 2018



# Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor David Arroyo el esfuerzo y la ayuda que ha desempeñado en esta tarea de aprendizaje.

A la universidad por la oportunidad de poder trabajar con ellos y el acceso al material necesario que me han ofrecido para poder solucionar los problemas encontrados.

A mi familia, por la confianza que han mostrado en mí. No solo por el apoyo económico, sin vuestro apoyo moral no habría llegado a ser quien soy hoy en día.

A Marta, que ha tenido que aguantar mis largas horas enfrente de un ordenador y aun así ha entendido que no solo sirve para jugar y mirar páginas de internet. Gracias por estar ahí siempre sin dudar.

Iago, no podría meterte en el grupo de amigos y dejarte ahí. Este año hemos estado más unidos que nunca y por eso te mereces una mención especial. Por las largas noches de insomnio y los abrazos aguantados. ¡Aún te debo una palmera!

Sanson...tu tampoco te salvas!! Fuiste la primera persona con la que hable en una universidad, recuerdo ese día en el descanso cuando con mi timidez me acerque a ese chico de pelo rizado con un iPhone 4 en la mano para preguntarle si la siguiente clase también sería ahí. El negocio fue que tú me enseñabas matemáticas y yo a ti programar, pero eso evoluciono en paseos por la 24h pidiendo que nos guardaran los portátiles y charlas de motos, sobre todo charlas de motos. Te deseo lo mejor, y deseo poderlo ver y que esta amistad no se apague con los nuevos retos que nos esperan.

A mi amigo Daniel, a ti siempre te nombraré hasta en los peores cabreos. Eres mi amigo de la infancia, el que no tiene ningún secreto. No has dudado ni un minuto en y siempre has mostrado admiración hacia mí, cosa que ha sido reciproca. Tú has encaminado mi madurez durante todo este tiempo.

Otra persona a la que me veo obligado a nombrar es Cintia, apenas nos conocemos desde hace menos de tres años. Eres la más nueva de esta larga lista, pero te lo has ganado! En poco tiempo has visto como soy un despistado para las conversaciones, y sin apenas broncas! Te agradezco toda la ayuda y apoyo. Y por todos los cotilleos que han acompañado esos viernes de comida después de trabajar.

Para finalizar, pero no menos importantes, gracias a todos mis amigos. Este año nos hemos visto menos, pero espero que volvamos a reunirnos con más frecuencia y no dejarlo en un buen recuerdo. Confío en ello y sé que lo haremos.



# Abstract

**Abstract** Over the last years, the use of Information and Communication Technologies (ICT) has grown significantly. As a consequence, many companies have decided to adapt their business according to the possibilities and needs of the digital world. This transition has been fueled by the irruption of cloud-based storage and computing means, along with the popularization of low cost smartphones. Nonetheless, this ecosystem conveys not only positive outcomes, but it also encloses some risks for the security of information systems. In fact, during the last years the number of attacks against web systems and applications has increased a lot. These attacks are anonymous and sometimes very difficult to be detected. Therefore, the hardening of these applications is a great challenge.

Web application development has evolved radically from server-based architectures to user-centered solutions. The objective of this project is to analyze the most critical security risks in the client-side development, what incorporates new types of attacks that did not exist in conventional systems built upon clear distinction between backend and frontend sub-systems. In addition, this project is aligned with the current trend in ICT: end users ubiquity and information access through smartphones. This work incorporates the necessary functionality to foster a secure and easy-to-use browsing experience.

This project studies mobile cloud applications vulnerabilities. Such a study is an important part of the design and development of a web application intended to manage the stock and the logistics of a company. This application considers a client that needs to manage different shops and warehouses. The project uses hybrid technologies based on a web environment that can be run in any device with any operating system. This software solution has been designed considering all the vulnerabilities studied here and adopting the adequate means to impede the corresponding security threats.

**Key words** — JavaScript, Node.js, Ionic, Framework, API REST, Json, hybrid application





# Resumen

## *Resumen*

En los últimos años ha tenido lugar un incremento muy significativo del uso de las Tecnologías de la Información y de las Comunicaciones (TICs). Esto ha provocado un gran incremento en el interés de las empresas por estar *online* y prestar sus servicios a través de plataformas digitales, actualizando sus herramientas de trabajo y digitalizando al máximo sus medios de producción. Este proceso de digitalización se ha visto potenciado por la irrupción de los servicios de almacenamiento y gestión en la nube, así como por la proliferación de dispositivos móviles inteligentes de bajo coste. Ahora bien, esta mejora en el acceso y manejo de nuestros datos no está exenta de riesgos ya que evitar los ciberataques supone a día de hoy un gran reto para cualquier proyecto. Los ataques digitales tienen cierta ventaja gracias al alcance y anonimato del atacante.

Con el uso de los servicios en la nube el desarrollo de aplicaciones web ha experimentado una evolución desde arquitecturas centradas en el servidor hacia soluciones centradas en el usuario. El objetivo de este trabajo es estudiar las principales amenazas que se pueden encontrar en aplicaciones web desarrolladas mediante enfoques centrados en el lado del cliente (*client-side web development*), lo que incorpora nuevos vectores de amenazas distintos a los sistemas convencionales (*backend* y *frontend*). Asimismo, el presente proyecto da cuenta de la otra gran tendencia existente en el ecosistema TIC: la ubicuidad de los usuarios finales y el acceso a los recursos mediante dispositivos móviles inteligentes. De modo correspondiente, el trabajo incorporará la funcionalidad necesaria para que los usuarios puedan utilizar la aplicación web desde sus dispositivos de modo seguro y favoreciendo una buena experiencia de navegación.

El presente proyecto estudia las vulnerabilidades de aplicaciones *cloud* móviles. Ahora bien, tal estudio se efectúa como parte en el diseño e implementación de una aplicación web con la que controlar una empresa de mercancías y su logística. Dicha aplicación ha sido solicitada por un cliente que necesita conocer la logística de las mercancías que compra a diario para diferentes tiendas con el fin de favorecer su adaptabilidad y el uso en diferentes sistemas. Se ha optado por el desarrollo de una aplicación híbrida, desarrollada utilizando tecnologías web que pueden ser ejecutadas en cualquier entorno móvil sin importar su sistema operativo. Toda la solución software ha sido diseñada considerando las vulnerabilidades estudiadas a lo largo del trabajo, de forma que en la implementación se han tenido en cuenta las principales amenazas a la seguridad de aplicaciones web desarrolladas en el lado del cliente, implementándose en cada caso las contramedidas necesarias.

**Palabras clave** — JavaScript, Node.js, Ionic, Framework, API REST, Json, aplicaciones híbridas



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.2. Objetivos . . . . .	3
1.3. Estructura del documento . . . . .	3
<b>2. Conceptos y Tecnologías</b>	<b>5</b>
2.1. Seguridad Informática . . . . .	5
2.2. JavaScript . . . . .	6
2.3. Node.js . . . . .	7
2.4. Aplicaciones Híbridas . . . . .	8
2.5. Bases de Datos no Relacionales . . . . .	9
2.6. REST . . . . .	10
<b>3. Aplicación</b>	<b>13</b>
3.1. Análisis del problema . . . . .	13
3.2. Usuarios . . . . .	14
3.3. Casos de uso . . . . .	15
3.3.1. Acciones . . . . .	15
3.4. Requisitos . . . . .	20
3.4.1. Requisitos Funcionalidades . . . . .	20
3.4.2. Requisitos No Funcionales . . . . .	21
3.5. Modelo Base de Datos . . . . .	22
3.6. API REST . . . . .	23
3.7. Cliente . . . . .	23
3.7.1. Mapa de navegación . . . . .	24
3.7.2. Interfaz de usuario . . . . .	24
3.7.3. Lectura de códigos de barras . . . . .	26
3.8. Pruebas unitarias y TDD . . . . .	26
<b>4. Principales Vulnerabilidades</b>	<b>29</b>
4.1. Seguridad en MongoDB . . . . .	29
4.2. Comunicaciones y Disponibilidad . . . . .	31
4.3. Autenticación . . . . .	35
4.3.1. Autenticación Básica . . . . .	35
4.3.2. Autenticación Avanzada . . . . .	38
4.4. Exposición de datos sensibles . . . . .	39
4.5. XXE . . . . .	40
4.6. XSS . . . . .	41

4.7. CSRF . . . . .	42
4.8. Configuraciones Incorrectas . . . . .	43
4.9. Versiones Antiguas . . . . .	45
<b>5. Conclusiones y Trabajo Futuro</b>	<b>47</b>
5.1. Conclusiones . . . . .	47
5.2. Trabajo Futuro . . . . .	48
<b>A. API REST</b>	<b>49</b>
<b>B. Prototipo Cliente</b>	<b>71</b>
B.1. <i>Login</i> y pantalla principal . . . . .	71
B.2. Inicio . . . . .	72
B.3. Crear/Modificar una entrada . . . . .	73
B.4. Usuarios . . . . .	74
B.5. Empresas . . . . .	75
B.6. Proveedores . . . . .	76
B.7. Categorías . . . . .	77
B.8. Productos . . . . .	78
B.9. Ubicaciones . . . . .	79
B.10. Ajustes . . . . .	80
B.11. Autenticación . . . . .	81

# Índice de tablas

3.1. Caso de uso: Hacer <i>Login</i> . . . . .	15
3.2. Caso de uso: Crear/Editar/Eliminar Productos. . . . .	15
3.3. Caso de uso: Crear/Editar/Eliminar Categorías. . . . .	16
3.4. Caso de uso: Crear/Editar/Eliminar Ubicaciones. . . . .	16
3.5. Caso de uso: Crear/Editar/Eliminar Compras. . . . .	16
3.6. Caso de uso: Modificar contraseña. . . . .	17
3.7. Caso de uso: Crear nueva empresa. . . . .	17
3.8. Caso de uso: Eliminar una empresa. . . . .	17
3.9. Caso de uso: Editar una empresa (Usuario administrador de aplicación). . .	18
3.10. Caso de uso: Editar una empresa (Usuario administrador de empresa). . .	18
3.11. Caso de uso: Crear/Eliminar usuarios (Usuario administrador de aplicación). .	18
3.12. Caso de uso: Crear/Eliminar usuarios (Usuario administrador de empresa). .	19
3.13. Caso de uso: Asignar un usuario a una empresa. . . . .	19
3.14. Caso de uso: Crear/Editar/Eliminar Proveedores (Administrador de aplicación). . . . .	19
3.15. Caso de uso: Crear/Editar/Eliminar Proveedores (Administrador de empresa). . . . .	20
A.1. API REST: POST /login. . . . .	49
A.2. API REST: GET /app/companies. . . . .	50
A.3. API REST: GET /app/company/:companyId. . . . .	50
A.4. API REST: POST /app/company. . . . .	51
A.5. API REST: PUT /app/company/:companyId. . . . .	51
A.6. API REST: DELETE /app/company/:companyId. . . . .	52
A.7. API REST: GET /app/providers. . . . .	52
A.8. API REST: GET /app/provider/:providerId. . . . .	53
A.9. API REST: POST /app/provider. . . . .	53
A.10. API REST: PUT /app/provider/:providerId. . . . .	54
A.11. API REST: DELETE /app/provider/:providerId. . . . .	54
A.12. API REST: GET /app/categories. . . . .	55
A.13. API REST: GET /app/categories/actives. . . . .	55
A.14. API REST: GET /app/category/:categoryId. . . . .	56
A.15. API REST: POST /app/category. . . . .	56
A.16. API REST: PUT /app/category/:categoryId. . . . .	57
A.17. API REST: DELETE /app/category/:categoryId. . . . .	57
A.18. API REST: GET /app/products. . . . .	58
A.19. API REST: GET /app/products/actives. . . . .	58
A.20. API REST: GET /app/products/category/:categoryId. . . . .	59

A.21.API REST: GET /app/product/:productId. . . . .	59
A.22.API REST: POST /app/product. . . . .	60
A.23.API REST: PUT /app/product/:productId. . . . .	60
A.24.API REST: DELETE /app/product/:productId. . . . .	61
A.25.API REST: GET /app/locations. . . . .	61
A.26.API REST: GET /app/locations/actives. . . . .	62
A.27.API REST: GET /app/location/:locationId. . . . .	62
A.28.API REST: POST /app/location. . . . .	63
A.29.API REST: PUT /app/location/:locationId. . . . .	63
A.30.API REST: DELETE /app/location/:locationId. . . . .	64
A.31.API REST: GET /app/orders. . . . .	64
A.32.API REST: GET /app/order/:orderId. . . . .	65
A.33.API REST: POST /app/order. . . . .	65
A.34.API REST: PUT /app/order/:orderId. . . . .	66
A.35.API REST: DELETE /app/order/:orderId. . . . .	66
A.36.API REST: GET /app/users. . . . .	67
A.37.API REST: GET /app/user/:userId. . . . .	67
A.38.API REST: POST /app/user. . . . .	68
A.39.API REST: GET /app/user. . . . .	68
A.40.API REST: PUT /app/user/:userId. . . . .	69
A.41.API REST: DELETE /app/user/:userId. . . . .	69
A.42.API REST: PUT /app/updatePassword. . . . .	70

# Índice de figuras

1.1. Número de módulos por lenguaje de programación. . . . .	2
1.2. TIOBE Index [5]. . . . .	3
2.1. Event Loop Node.js (Fuente [11]). . . . .	8
2.2. Arquitectura REST. . . . .	10
3.1. Diagrama de casos de uso. . . . .	14
3.2. Modelo Base de datos. . . . .	23
3.3. Mapa de navegación. . . . .	24
3.4. Lectura de códigos de barra. . . . .	26
3.5. Test de integración: <i>Login</i> . . . . .	27
4.1. Ataque NoSQL Injection. . . . .	30
4.2. Protección NoSQL Injection. . . . .	30
4.3. POST <i>Login</i> sin HTTPs. . . . .	31
4.4. <i>Tokens</i> en las cabeceras de las peticiones. . . . .	31
4.5. Ataque Man in the Middle. . . . .	32
4.6. Generación claves publicas/privadas. . . . .	33
4.7. Crear un servidor Node.js con conexión HTTPS. . . . .	33
4.8. Configuración prevención ataques DDoS. . . . .	34
4.9. Respuesta ante ataque DDoS. . . . .	34
4.10. Creación y validación de <i>tokens</i> de sesión. . . . .	37
4.11. <i>Middleware Login</i> . . . . .	37
4.12. Control de acceso por <i>middlewares</i> . . . . .	38
4.13. Cifrado de contraseñas con bCrypt. . . . .	39
4.14. Protección de datos sensibles. . . . .	40
4.15. Protección XSS Ionic. . . . .	42
4.16. Código parseado XSS. . . . .	42
4.17. Esquema Ataque CSRF. . . . .	43
4.18. Configuración en MongoDB. . . . .	44
4.19. Control de acceso a MongoDB. . . . .	44
4.20. Vulnerabilidades en versiones antiguas. . . . .	46
4.21. Actualización a la última versión. . . . .	46
B.1. Inicio de sesión. . . . .	71
B.2. Inicio. . . . .	72
B.3. Crear/Modificar una entrada. . . . .	73
B.4. Usuarios. . . . .	74
B.5. Empresas. . . . .	75

B.6. Proveedores. . . . .	76
B.7. Categorías. . . . .	77
B.8. Productos. . . . .	78
B.9. Ubicaciones. . . . .	79
B.10.Ajustes. . . . .	80
B.11.Cambiar contraseña/token doble paso. . . . .	81



# 1 | Introducción

La evolución hacia la digitalización de los servicios ha generado novedades en términos de arquitecturas como *serverless* y con ello, ha provocado un crecimiento de las amenazas con nuevos vectores de ataque y mayor volumen. Este Trabajo de Fin de Máster tiene como propósito el análisis de la seguridad y vulnerabilidad en aplicaciones web basadas en Node.js y MongoDB. Para ello, se realizará el desarrollo de una aplicación cliente-servidor<sup>1</sup>, la cual será implantada en un entorno real donde la parte más fuerte de la misma será la propia seguridad. El conjunto de tecnologías empleadas en este proyecto se adscriben a la tendencia de evolución desde aplicaciones web cuyo desarrollo está centrado en el servidor, hacia soluciones web en las que el diseño está centrado en el cliente. Este nuevo paradigma tiene por horizonte último las denominadas arquitecturas *serverless*, y tiene asociado un conjunto de problemas (en términos de funcionalidad y de seguridad) distinto del asociado a las aplicaciones web convencionales con una clara diferencia entre sistema *backend* y *frontend*. En este capítulo se detalla una breve descripción de los motivos que han llevado a la realización de este proyecto, así como sus objetivos principales, seguidos de la estructura del presente documento.

## 1.1. Motivación

En los últimos años el uso de las tecnologías basadas en la nube ha crecido de manera vertiginosa. Esto ha supuesto un gran reto en el desarrollo de tecnologías web, así como un objetivo a atacar para aquellos que quieren cometer un delito como robar información o generar un fallo en el sistema. El fácil acceso y anonimato aportan ventajas a los atacantes por el número de ataques y falta de riesgos físicos, lo que supone un reto para el desarrollador por la cantidad y variedad de posibles ataques que puede sufrir.

Los modelos convencionales de seguridad, como los cortafuegos o antivirus, no protegen a los sistemas de las vulnerabilidades a nivel de aplicación [1]. El desarrollo de una aplicación segura, con el actual modelo de programación web, es realmente complejo debido a la facilidad de cometer un pequeño error de seguridad provocando el acceso a datos sensibles o control al propio sistema.

El número de ataques a sistemas web aumentan cada día, sólo en 2017 se bloquearon una media de 611.000 ataques cada día [2]. Estos ataques han aumentado tanto en cantidad como en variedad. Este incremento en la modalidad de los ataques viene, en cierto modo, derivado por una tendencia a hacer cada vez menos diferenciable la parte de servidor de la parte de cliente de las aplicaciones web. En efecto, si en los inicios de la tecnología

---

<sup>1</sup>Diseño de software cuyo objetivo es centrar toda la lógica de negocio sobre un servidor central que realiza acciones por demanda de terceras partes, los clientes

AJAX, se empleaba elementos como JavaScript para mejora la experiencia de usuario interviniendo exclusivamente en la parte cliente del servidor, cada vez es más habitual que estas tecnologías de cliente se empleen también para actuar directamente sobre la sección de servidor de las aplicaciones [3], [4].

Por otro lado, JavaScript está evolucionando notablemente en estos últimos años, tanto en cantidad de *frameworks* como en bibliotecas disponibles. Npm es el mayor proveedor de módulos que existe actualmente. Si lo comparamos con otros lenguajes como .Net, Python o Ruby (figura 1.1). Npm ha crecido de forma exponencial frente al crecimiento lineal del resto módulos.

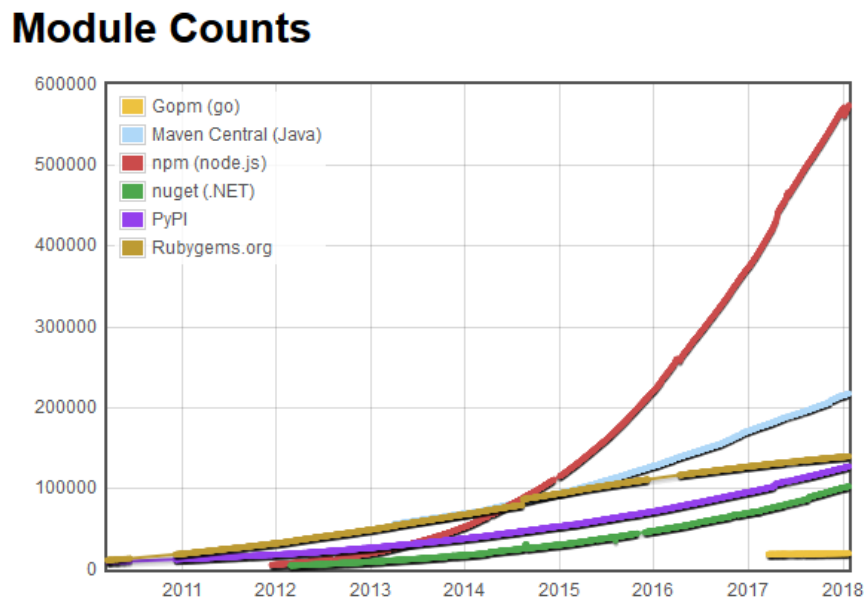


Figura 1.1: Número de módulos por lenguaje de programación.

Si miramos el ranking de lenguajes de programación más utilizados (figura 1.2), JavaScript se encuentra en el top 10. Estos números lo hacen un lenguaje muy interesante, tanto para el desarrollador por la cantidad de desarrollo e información que puede encontrar, como para el atacante por el alcance que puede tener al encontrar una vulnerabilidad.

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215%	-3.06%
2	2		C	11.037%	+1.69%
3	3		C++	5.603%	-0.70%
4	5	▲	Python	4.678%	+1.21%
5	4	▼	C#	3.754%	-0.29%
6	7	▲	JavaScript	3.465%	+0.62%
7	6	▼	Visual Basic .NET	3.261%	+0.30%
8	16	▲▲	R	2.549%	+0.76%
9	10	▲	PHP	2.532%	-0.03%
10	8	▼	Perl	2.419%	-0.33%

Figura 1.2: TIOBE Index [5].

## 1.2. Objetivos

El objetivo principal de este proyecto es el análisis de las vulnerabilidades que más sufren las aplicaciones basadas en tecnologías web como Node.js y MongoDB [3].

Para ello desarrollaremos un sistema de control de mercancías con el que realizaremos las pruebas de seguridad más adelante. El sistema que implementaremos estará dividido en *backend*, un servidor API REST implementado con la tecnología Node.js apoyado con una base de datos en MongoDB, y *frontend*, utilizando una tecnología híbrida cuyo objeto es la posible ejecución de este desde cualquier sistema móvil. Para ello utilizaremos el *framework* Ionic [6], que se basa en la tecnología Angular y que favorece una interacción activa entre usuario y sistema, y en la biblioteca Cordova<sup>2</sup> para su compilación creación de aplicaciones híbridas para distintos dispositivos móviles como puede ser IOS y Android.

El desarrollo del proyecto se ha inspirado en las propuestas Webgoat<sup>3</sup> y Nodegoat<sup>4</sup> de la iniciativa OWASP [7]. Así, la aplicación se diseñará e implementará sin contemplar las posibles vulnerabilidades que son objetivo de estudio. Una vez desarrollada, se analizarán dichas vulnerabilidades y se realizará una batería de pruebas en la aplicación para analizar los resultados. Tras este análisis se realizarán las acciones necesarias para mejorar la seguridad en la aplicación y se repetirán las pruebas para evaluar las diferencias.

## 1.3. Estructura del documento

El trabajo de estudio de amenazas de seguridad y el despliegue de las contramedidas oportunas se efectuará de acuerdo al siguiente esquema:

<sup>2</sup><https://cordova.apache.org/>. Último Acceso 10 de junio de 2018.

<sup>3</sup><https://github.com/WebGoat/WebGoat>. Último Acceso 10 de junio de 2018.

<sup>4</sup><https://github.com/OWASP/NodeGoat>. Último Acceso 10 de junio de 2018.

- En el **capítulo 2** se habla sobre los conceptos generales sobre seguridad en la red y las herramientas existentes para el desarrollo de la misma.
- En el **capítulo 3** se tratará la aplicación a desarrollar, mostraremos el problema a solventar y la solución propuesta. Así como su diseño, casos de uso y desarrollo.
- En el **capítulo 4** se realiza un estudio de las principales vulnerabilidades web, por cada uno se realizará un ejemplo sencillo de prueba para ver cómo afecta dicha vulnerabilidad a la aplicación, y en el caso de que afecte, se definirán las medidas oportunas para prevenirla.
- En el **capítulo 5** se desarrollarán los test de integración que garanticen el correcto funcionamiento de la aplicación.
- Acabará el **capítulo 6** con las conclusiones obtenidas a lo largo del proyecto y futuros trabajos derivados del mismo.

## 2 | Principales conceptos y tecnologías utilizadas a lo largo del proyecto

Una vez definido el motivo y los objetivos del trabajo y antes de enfocarnos directamente sobre el problema, es necesario saber la situación actual del mismo. Comprender el objetivo, las tecnologías y arquitecturas utilizadas, es esencial para conocer el contexto del problema. En este capítulo hablaremos sobre los conceptos principales que se van a tratar a lo largo del proyecto.

### 2.1. Seguridad Informática

Siguiendo la definición de John Vacca, la seguridad informática es una rama de la informática y está enfocada a la protección del *hardware/software* los sistemas y redes [8]. Esto hace referencia a todos los medios utilizados para resguardar y proteger las comunicaciones, el tratamiento y almacenamiento de la información. La seguridad informática se basa en cuatro vértices que garantizan el correcto funcionamiento de una aplicación. Estos vértices son:

- **Confidencialidad:** Sólo los usuarios que tienen permiso a los datos deben tener acceso a los mismos. De forma general, la propiedad de confidencialidad se consigue mediante el uso adecuado de la criptografía (simétrica y asimétrica).
- **Integridad:** La aplicación debe garantizar que la información que recibe el usuario es verdadera y completa. Las funciones *hash* son los principales mecanismos de verificación de integridad de información.
- **Disponibilidad:** La aplicación debe estar disponible sin caídas ni retrasos inesperados. Este objetivo, pues, requiere el despliegue de medidas de redundancia de diversa naturaleza.
- **Autenticación:** La aplicación debe demostrar que es quien dice ser, evitando que otra persona pueda robar su identidad para interactuar con el usuario. Esto se consigue mediante el uso de claves públicas/privadas y entidades certificadoras que garanticen la veracidad de dichas claves. En el contexto actual, además, se establecen mecanismos que permiten verificar la validez de las entidades o autoridades de certificación [9].

Estos criterios son condiciones básicas (necesarias pero no suficientes) para poder conseguir cierto nivel de seguridad.

### 2.2. JavaScript

JavaScript es un lenguaje de programación nacido para la ejecución de acciones en los navegadores web, consiguiendo la creación de páginas con más dinamismo y mejor respuesta a las acciones del usuario. Este lenguaje nació en 1995 con el nombre de Mocha creado por Brendan Eich, empleado de Netscape. Más tarde se cambió de nombre por el de LiveScript hasta finalmente en diciembre de 1995 se le llamo JavaScript como truco de *marketing* al asociarse con el lenguaje de programación Java, con el que no comparten ninguna relación.

En marzo de 1996, Netscape lanzó el navegador Navigator 2.0 al que le incorporo soporte para JavaScript. Ante este lanzamiento Microsoft desarrollo un clon para su navegador Internet Explorer 3 al que le llamó JScript. Para evitar una guerra de patentes tecnológicas, Netscape decidió presentar JavaScript ante el organismo ECMA (European Computer Manufacturers Association) para estandarizar el lenguaje, creando el estándar ECMAScript. Al principio este lenguaje solo servía para validar ciertos campos y pequeñas funciones, pero gracias a un rápido progreso pronto se lanzaron las versiones ES2 y ES3 agregando funciones más críticas como XMLHttpRequest, funciones muy importantes para la posterior popularización de AJAX.

Entre el 2005 y 2008 hubo un gran número de enfrentamientos sobre la compatibilidad con versiones anteriores, lo que provocó que se crearan dos versiones del estándar naciendo ES3.1 y ES4. Finalmente, el organismo encargado de tomar las decisiones finales decidió abandonar el estándar ES4 y en diciembre del 2009 se lanzó ES5. Para las siguientes versiones faltaron menos años presentándose ES6 en junio de 2015 bajo el nombre ES2015 y a partir de este se hacen presentaciones anuales con las que se añaden nuevas características al lenguaje garantizando siempre la compatibilidad con versiones anteriores.

En este momento, JavaScript es el lenguaje más popular de Github teniendo el mayor número de repositorios[10]. Uno de los factores de su éxito se debe al lanzamiento por parte de Google de Gmail en el 2004. Con Gmail Google creo un nuevo tipo de aplicación donde no era necesario recargar una página por completo para actualizar pequeñas partes dentro de la misma. Fue la primera web de una sola página. Esta técnica de refresco mediante el uso de XMLHttpRequest fue conocida como AJAX. Gracias a Gmail, Google comprendió que los futuros navegadores web necesitarían JavaScript para la creación de aplicaciones web que mejoraran la experiencia del usuario. Fue con esto con lo que creo en motor V8, de código abierto con el que se impulsa Chrome y más adelante aparecieron nuevos entornos como Node.js como necesidad de transportar las ventajas de JavaScript a entornos fuera del navegador.

Para manipular el DOM de la web, manejar eventos y tener mejor integración con AJAX, en el 2006 se lanzó JQuery. Esto mejoro la interacción con el usuario, pero generó mucho código "Espagueti" hasta el 2010 con la aparición de backbone.js con el fin de luchar contra este código. Esta biblioteca proponía una estructura MVC en las aplicaciones web. A su vez, Google empezó a desarrollar Angular.js. Una biblioteca que permitiría generar

aplicaciones web complejas de una sola página. No consiguió gran popularidad hasta principios del 2013 y en 2014 anunciaron una nueva versión con Angular 2, el cual no era compatible con Angular.js y se le añadían nuevas características como TypeScript. TypeScript es un superconjunto de JavaScript que añade tipado estático y clases. Este lenguaje está pensado para poder ser compilado en a código JavaScript consiguiendo el desarrollo de grandes aplicaciones compatibles con los navegadores.

A partir de esto y con el aumento del uso del teléfono móvil aparecen las primeras aplicaciones híbridas. Aplicaciones web que son capaces de ser ejecutadas en teléfonos móviles, esto es gracias a *frameworks* como el de Apache Cordova que genera un navegador web con el motor V8 de Google para ejecutar la aplicación web desarrollada. Al ser un lenguaje del lado del cliente, sus limitaciones son mayores frente a otros lenguajes como C, que ataca directamente sobre el *core* del sistema que lo ejecuta.

Su buena respuesta por lado del navegador ha conseguido popularizarlo estando a día de hoy en el top 10 de lenguajes de programación según la lista TIOBE (figura 1.2). Gracias a esto y a su gran comunidad de desarrolladores han creado un conjunto de librerías, con lo que se puede cubrir casi cualquier funcionalidad. Una de las librerías más grandes y potentes es Node.js, desarrollada para la ejecución de código JavaScript en el lado del servidor, posee uno de los proveedores de paquetes más grandes en cuanto a módulos de librerías. A continuación, trataremos más sobre esta librería en concreto.

## 2.3. Node.js

Node.js es un entorno de ejecución multiplataforma cuyo objetivo es ejecutar código JavaScript en la parte del servidor. Esto lo hace gracias a su intérprete JavaScript V8, motor desarrollado en C++ por Google para su navegador Chrome. Este motor tiene la característica de que puede ser descargado en cualquier aplicación y ejecutado desde la misma sin la necesidad de un navegador.

Uno de sus principales fuertes frente a otros lenguajes es su ejecución asíncrona dirigida a eventos. Frente a otros lenguajes que esperan una llamada y generan un hilo por petición, con su reserva de memoria y tiempo de espera, limitando en número de usuarios conectados a la vez. Node.js utiliza un modelo de programación mono-hilo dirigido a eventos. Exactamente Node.js funciona de la siguiente manera:

Node trabaja con un único hilo que es el encargado en repartir todas las peticiones que van llegando. Para poder ejecutar todas las peticiones, delega todo el trabajo a un pool de hilos. Una vez uno de estos hilos ha realizado el trabajo realizado emite un evento al hilo principal, recibido este evento una función *callback* se encargará de procesar el resultado. Esto hace que Node siempre esté preparado para recibir eventos de forma asíncrona con lo que ahorramos el tiempo y la memoria necesaria para la creación de hilos y los bloqueos esperando la respuesta de la ejecución de ciertos códigos (figura 2.1).

Otro de sus principales fuertes es su sistema de paquetes (Node Package Manager), es el mayor sistema de paquetes que existe actualmente en los lenguajes de programación. Su

licencia de uso Open Source ha creado una gran comunidad de desarrollo que ha apoyado al crecimiento del este.

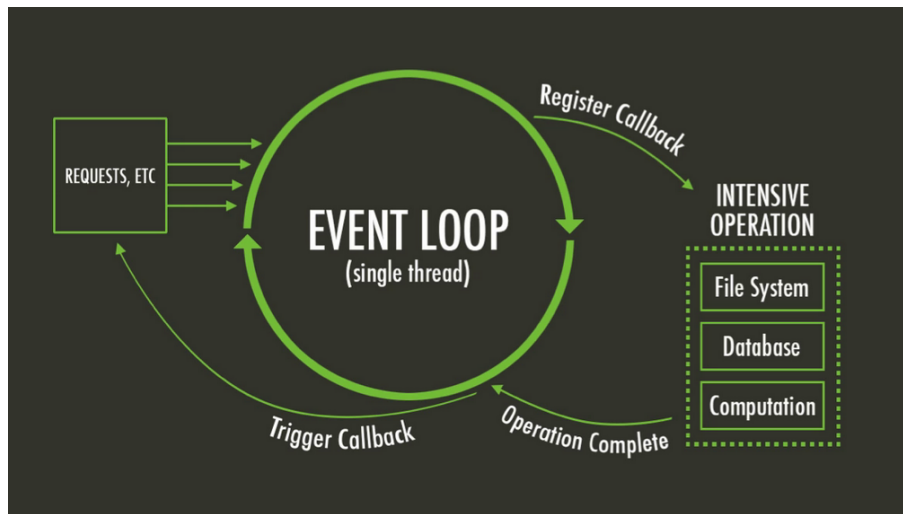


Figura 2.1: Event Loop Node.js (Fuente [11]).

## 2.4. Aplicaciones Híbridas

El elevado ritmo de desarrollo de aplicaciones móviles y el número de plataformas diferentes, obligaban tener varios equipos de trabajo especializados en una plataforma concreta, lo que supone un sobre coste que no todas las empresas pueden asumir, y menos las empresas emergentes que empiezan.

Las aplicaciones híbridas son aquellas que han sido desarrolladas utilizando tecnologías web como HTML5, JavaScript y CSS que se ejecutan en el navegador nativo del sistema operativo (S.O.) y no en el propio *core*. Consiguiendo así que sea indiferente el S.O. en el que se ejecutan dejando esa dependencia en manos del navegador del propio sistema.

Las principales ventajas de estas aplicaciones frente a las nativas es el coste de desarrollo y la reutilización del código. Frente a esto, la principal ventaja de las nativas radica en la ejecución directa sobre el *core*, lo que supone un mayor rendimiento y acceso a características especiales del *hardware*, lo que se está reduciendo debido al gran número de librerías que están apareciendo para poder utilizar estas características y la optimización de código de estos *frameworks*.

Para el desarrollo de este proyecto no es necesario la realización de un cómputo complejo por parte del cliente, ni el acceso a características del *hardware* que nos obliguen al uso de aplicaciones nativas. Debido a esto, para tener un mayor alcance se desarrollará una aplicación híbrida.

Entre las principales opciones que se pueden encontrar son las siguientes:

- **Ionic:** Es de los *frameworks* más populares, está integrado con Angular y se ejecuta



en una *app* basada en Cordova para ser ejecutado en los diferentes dispositivos móviles. Uno de sus principales puntos fuertes es su línea de comandos, la cual proporciona una gran variedad de características. Entre ellas emuladores de S.O. integrados.

- **Framework 7:** El potencial de este *framework* es la no dependencia de otros *frameworks* como pueden ser Angular o React, esto supone que para poder ser ejecutado es necesario la combinación de este con Cordova o PhoneGap.
- **React Native:** Es un *framework* desarrollado por Facebook, cuyo objetivo es la traducción de aplicaciones JavaScript a lenguajes nativos como Android u Objective-C. Su mayor limitación es su curva de aprendizaje, que puede ser apoyada gracias a su gran comunidad y documentación.
- **JQuery Mobile:** JQuery, a diferencia del resto de *frameworks*, no está pensado para simular una aplicación nativa. El objetivo de este *framework* es el desarrollo de aplicaciones web que funcionan en todos los navegadores (incluidos los antiguos).
- **Native Script:** Al igual que React Native, el objetivo de este *framework* es la traducción al lenguaje nativo.

## 2.5. Bases de Datos no Relacionales

Las bases de datos no relacionales, también conocidas como NoSQL (Not Only SQL), son la solución que se encontró para solventar el problema de escalabilidad y rendimiento que tienen las bases de datos convencionales al trabajar con toda la información que genera la web. Al llegar la web 2.0, las bases de datos relacionales se encontraron con un problema de estructura y escalabilidad. Los datos que se generaban provenían de diferentes fuentes con estructuras no definidas. Esto generaba un gran problema al crear un modelo de bases de datos que cubra todas las posibilidades. Las bases de datos NoSQL son bases de datos que guardan información sin una estructura fija y sin relacionar una información con otra (**JOINS**), principalmente sobre documentos de forma clave valor, documentales o grafos de datos. Sus principales ventajas frente a las bases de datos relacionales son:

- **No necesitan muchos recursos:** No requieren de una computación compleja y de alto rendimiento para obtener rápidos resultados.
- **Sistema de almacenamiento distribuido:** Los sistemas de almacenamiento se distribuyen mediante nodos repartiéndose la información con copias de seguridad para que sea fiable.
- **Trabajan con un gran volumen de datos:** Gracias a su estructura distribuida pueden trabajar con un volumen de datos superior al de las bases de datos convencionales consiguiendo rápidos resultados mediante técnicas como MapReduce en la que cada nodo trabaja con una parte de la información hasta llegar los resultados al nodo principal.

- **Sin cuellos de botella:** Uno de los problemas que nos encontramos con las bases de datos relacionales y la cantidad de computo que es necesario para la asociación de tablas y la validación de acciones lo supone una alta complejidad de ejecución de ciertas sentencias. Para evitar esto no es recomendable el uso de JOINS en NoSQL.

## 2.6. REST

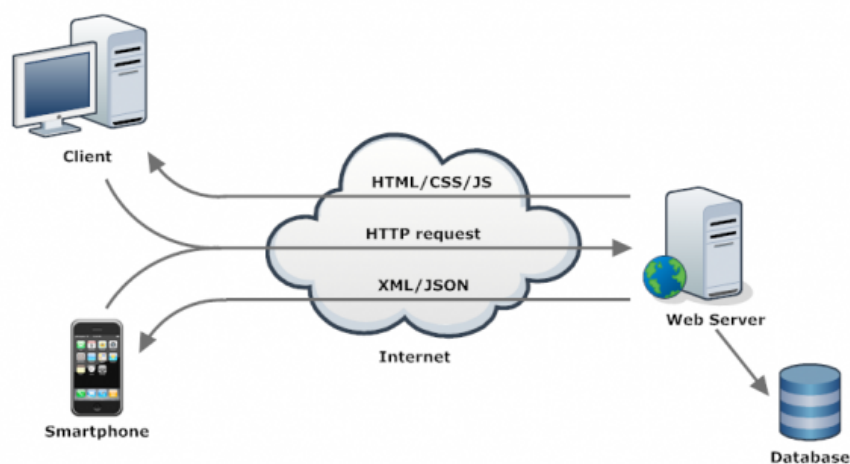


Figura 2.2: Arquitectura REST.

REST (Representational state transfer) es un estilo de arquitectura de *software* que se utiliza para la comunicación de sistemas distribuidos. Es una interfaz utilizada entre sistemas, que mediante el uso de HTTP transfieren datos o realizan operaciones (figura 2.2). Es una alternativa a otros protocolos de intercambio de datos como SOAP (Simple Object Access Protocol), protocolo de gran capacidad para la comunicación entre sistemas pero altamente complejo de interpretar por la propia máquina. Las principales características de REST son:

- **Protocolo cliente/servidor sin estado:** No se guarda ningún estado entre cliente y servidor, por lo que si dos clientes realizan la misma petición la respuesta será la misma sin necesidad de caché ni mantener una sesión iniciada.
- **Operaciones específicas de HTTP:** Las principales operaciones que se pueden ejecutar están especificadas por el protocolo HTTP. Las operaciones principales son: POST (crear), GET (consultar), PUT (editar) y DELETE (eliminar).
- **Todas las ejecuciones son a partir de una URI:** Se utiliza la URI como identificador único para la realización de operaciones.

- **Uso de hipermedios:** REST permite el uso de hipermedios para dar acceso a otros recursos, esta transferencia se hace siguiendo el principio HATEOAS. El servidor envía las direcciones donde se encuentra la información y el cliente va navegando por la misma sin necesidad de tener conocimiento sobre la forma de interactuar con la aplicación.



## 3 | Aplicación

### 3.1. Análisis del problema

La aplicación para el proyecto trata de solventar un problema que tiene un cliente para la gestión de mercancías en su comercio. El cliente ha detectado un problema al no controlar el destino de las mercancías. El cliente, que posee varias fruterías en Madrid norte, distribuye a diario a cada uno de los empleados una lista con el detalle de las compras necesarias para cada una de las tiendas. Mientras se realizan las compras, si ven productos con buen precio o estado pueden añadir dichos productos a la lista. También se da el caso de que un mismo producto se compre en varios puestos por volumen o precio, o que se modifique el volumen de compra de un producto.

En la actualidad, para tener un control de esta situación, se identifica cada caja y se avisa a los empleadores correspondientes con los puestos.

Ahora mismo, hay tres empleados comprando y dos cargando los productos en tres furgonetas para tres tiendas diferentes. Realizada la compra, una vez llegan los vehículos a las tiendas y se ha descargado la mercancía se suelen encontrar con los siguientes casos:

- **Reparto correcto:** Es el mejor de los casos, todas las mercancías han llegado a su destino y se ha hecho la compra correcta.
- **Compra excesiva:** Se han encontrado casos en los que un comprador no ha visto que un compañero suyo ha realizado la misma compra y duplican la compra. Este caso apenas se da ya que cada uno compra una sección de la "nota", y en caso de comprar algo del compañero avisa por el *walkie* y este lo tacha de la lista para saber que no lo tiene que comprar.
- **Los productos no llegan a la tienda que deben llegar:** Uno de los casos más frecuentes es que se varíen las cantidades de compra, se apunten en un papel el nuevo reparto, pero este difiere mucho de la realidad final. Este caso está suponiendo un sobrecoste innecesario para la empresa, ya que es frecuente que al menos un producto este mal repartido, lo que provoca que una vez se han descargado los camiones se tiene un empleado llevando los productos mal repartidos a su verdadero destino. Esto se resume en tener un empleado dos horas fuera de su puesto de trabajo, un vehículo circulando con el coste oportuno y un producto en el vehículo sin poder ser vendido.
- **Los productos no salen de Mercamadrid:** Este caso se da en pocas ocasiones. Para estos casos se pide al comercial que se lo guarde para el día siguiente en las

cámaras frigoríficas del mismo o, en caso de necesitar el producto para el mismo día, suele ir el propio dueño a Mercamadrid con su coche a por los productos.

## 3.2. Usuarios

La aplicación será utilizada por tres tipos de usuarios:

- **Administrador de aplicación:** Tiene control total sobre la aplicación. Tiene la capacidad de crear nuevas empresas, usuarios y asignar usuarios a dichas empresas. Este rol está pensado para la expansión de la aplicación pudiendo ser utilizada por diferentes clientes sobre el mismo servidor.
- **Administrador de empresa:** Tiene control total sobre una empresa en concreto. Puede crear nuevos usuarios de tipo administradores de empresa o empleados, y modificar los datos de la empresa. Crear proveedores y todas las funcionalidades que tenga el rol de usuario.
- **Empleado:** Es el usuario más limitado ya tiene las funcionalidades necesarias para poder realizar compras, crear productos, ubicaciones y categorías. Este rol no puede crear ni editar otros usuarios, ni modificar los datos de la empresa para la que trabaja ni crear/editar proveedores.

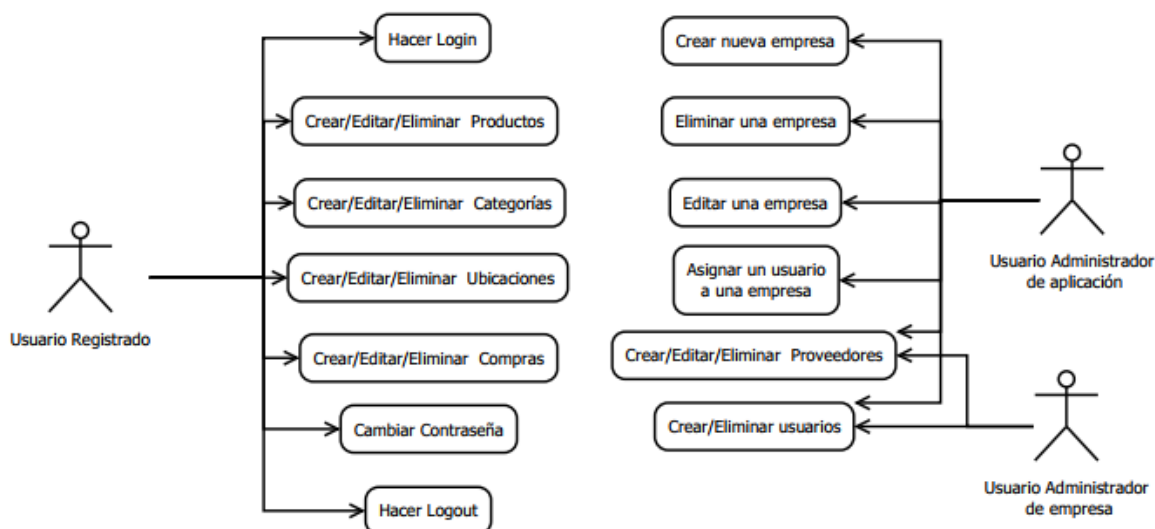


Figura 3.1: Diagrama de casos de uso.

### 3.3. Casos de uso

Los casos de uso se obtienen a partir de la toma de requisitos del usuario. Estos representan cada de las acciones que podrá realizar el usuario desde la interfaz.

#### 3.3.1. Acciones

A continuación, se describirán las acciones definidas en los casos de uso (ver figura 3.1), añadiendo las precondiciones que se deben dar para que se pueda realizar y los resultados obtenidos:

Hacer <i>Login</i>	
<b>Usuario</b>	Usuario Registrado
<b>Descripción</b>	
El usuario introduce su nombre de usuario y contraseña	
<b>Precondiciones</b>	El usuario debe estar dado de alta en la aplicación.
<b>Resultado</b>	La aplicación devuelve un <i>token</i> de sesión único con el que se identifica el usuario. La aplicación se dirige al menú principal.

Tabla 3.1: Caso de uso: Hacer *Login*.

Crear/Editar/Eliminar Productos	
<b>Usuario</b>	Usuario Registrado
<b>Descripción</b>	
El usuario crea, edita o eliminar productos de su empresa	
<b>Precondiciones</b>	El usuario debe estar <i>logueado</i> en el sistema.
<b>Resultado</b>	La aplicación ejecuta la acción del usuario devolviendo el estado del producto.

Tabla 3.2: Caso de uso: Crear/Editar/Eliminar Productos.

Crear/Editar/Eliminar Categorías	
<b>Usuario</b>	Usuario Registrado
Descripción	
El usuario crea, edita o eliminar categorías de su empresa	
<b>Precondiciones</b>	El usuario debe estar <i>logueado</i> en el sistema.
<b>Resultado</b>	La aplicación ejecuta la acción del usuario devolviendo el estado de la categoría.

Tabla 3.3: Caso de uso: Crear/Editar/Eliminar Categorías.

Crear/Editar/Eliminar Ubicaciones	
<b>Usuario</b>	Usuario Registrado
Descripción	
El usuario crea, edita o eliminar ubicaciones de su empresa	
<b>Precondiciones</b>	El usuario debe estar <i>logueado</i> en el sistema.
<b>Resultado</b>	La aplicación ejecuta la acción del usuario devolviendo el estado de la ubicación.

Tabla 3.4: Caso de uso: Crear/Editar/Eliminar Ubicaciones.

Crear/Editar/Eliminar Compras	
<b>Usuario</b>	Usuario Registrado
Descripción	
El usuario crea, edita o eliminar compras de su empresa	
<b>Precondiciones</b>	El usuario debe estar <i>logueado</i> en el sistema.
<b>Resultado</b>	La aplicación ejecuta la acción del usuario devolviendo el estado de la compra.

Tabla 3.5: Caso de uso: Crear/Editar/Eliminar Compras.



Cambiar contraseña	
Usuario	Usuario Registrado
Descripción	
El usuario modifica su contraseña	
Precondiciones	El usuario debe estar <i>logueado</i> en el sistema.
Resultado	La modifica la contraseña del usuario <i>logueado</i> .

Tabla 3.6: Caso de uso: Modificar contraseña.

Crear una nueva empresa	
Usuario	Usuario administrador de aplicación
Descripción	
El usuario introduce y crea una nueva empresa	
Precondiciones	El usuario debe estar <i>logueado</i> en el sistema y tener rol de administrador de la aplicación.
Resultado	La aplicación crea una nueva empresa y devuelve los datos de la misma.

Tabla 3.7: Caso de uso: Crear nueva empresa.

Eliminar una empresa	
Usuario	Usuario administrador de aplicación
Descripción	
El usuario elimina una empresa	
Precondiciones	El usuario debe estar <i>logueado</i> en el sistema y tener rol de administrador de la aplicación.
Resultado	La aplicación elimina la empresa y todos los datos asociados a la misma, usuarios de la empresa, productos, categorías, ubicaciones y compras.

Tabla 3.8: Caso de uso: Eliminar una empresa.

Editar una empresa	
Usuario	Usuario administrador de aplicación
Descripción	
El usuario edita una empresa	
Precondiciones	El usuario debe estar <i>logueado</i> en el sistema y tener rol de administrador de la aplicación.
Resultado	La aplicación modifica los datos de la empresa y devuelve el resultado final.

Tabla 3.9: Caso de uso: Editar una empresa (Usuario administrador de aplicación).

Editar una empresa	
Usuario	Usuario administrador de empresa
Descripción	
El usuario edita la empresa a la que pertenece	
Precondiciones	El usuario debe estar <i>logueado</i> en el sistema y tener rol de administrador de la empresa a editar.
Resultado	La aplicación modifica los datos de la empresa y devuelve el resultado final.

Tabla 3.10: Caso de uso: Editar una empresa (Usuario administrador de empresa).

Crear/Eliminar usuarios	
Usuario	Usuario administrador de aplicación
Descripción	
El usuario crea/elimina usuarios en la aplicación	
Precondiciones	El usuario debe estar <i>logueado</i> en el sistema y tener rol de administrador de la aplicación. El administrador debe añadir la empresa para la que trabaja el usuario.
Resultado	La aplicación crea/elimina el usuario devolviendo el resultado final.

Tabla 3.11: Caso de uso: Crear/Eliminar usuarios (Usuario administrador de aplicación).

Crear/Eliminar usuarios	
<b>Usuario</b>	Usuario administrador de empresa
<b>Descripción</b>	
El usuario crea/elimina usuarios en la aplicación	
<b>Precondiciones</b>	El usuario debe estar <i>logueado</i> en el sistema y tener rol de administrador de la empresa. El usuario a eliminar debe pertenecer a la empresa del administrador.
<b>Resultado</b>	La aplicación crea/elimina el usuario devolviendo el resultado final. En caso de crear, la aplicación asigna el usuario a la empresa a la que pertenece el administrador.

Tabla 3.12: Caso de uso: Crear/Eliminar usuarios (Usuario administrador de empresa).

Asignar un usuario a una empresa	
<b>Usuario</b>	Usuario administrador de aplicación
<b>Descripción</b>	
El administrador asigna un usuario a una empresa del sistema	
<b>Precondiciones</b>	El usuario debe estar <i>logueado</i> en el sistema y tener rol de administrador de aplicación.
<b>Resultado</b>	La aplicación el usuario a la empresa y devuelve los resultados.

Tabla 3.13: Caso de uso: Asignar un usuario a una empresa.

Crear/Editar/Eliminar Proveedores	
<b>Usuario</b>	Usuario administrador de aplicación
<b>Descripción</b>	
El usuario crea, edita o eliminar proveedores	
<b>Precondiciones</b>	El usuario debe estar <i>logueado</i> en el sistema y tener rol de administrador de la aplicación.
<b>Resultado</b>	La aplicación ejecuta la acción del usuario devolviendo el estado del proveedor. En caso de creación, el proveedor pertenecerá a la empresa a la que este asignado el usuario.

Tabla 3.14: Caso de uso: Crear/Editar/Eliminar Proveedores (Administrador de aplicación).

Crear/Editar/Eliminar Proveedores	
Usuario	Usuario administrador de empresa
Descripción	
El usuario crea, edita o eliminar proveedores de su empresa	
Precondiciones	El usuario debe estar <i>logueado</i> en el sistema y tener rol de administrador de la empresa.
Resultado	La aplicación ejecuta la acción del usuario devolviendo el estado del proveedor.

Tabla 3.15: Caso de uso: Crear/Editar/Eliminar Proveedores (Administrador de empresa).

## 3.4. Requisitos

Después de hablar con el usuario final, detectar los problemas a resolver y concretar los casos de uso, pasamos al proceso de definir los requisitos necesarios para cubrir todas las necesidades. Para ello, se dividirán los requisitos en funcionales y no funcionales. A continuación, se numerarán y definirán cada uno de ellos.

### 3.4.1. Requisitos Funcionalidades

- **[RF1] Registro de empresas:** Los usuarios con perfil de administrador de la aplicación podrán crear nuevas empresas.
- **[RF2] Registro de usuarios:** Los usuarios con perfil de administrador de la aplicación o administrador de empresa podrán crear nuevos usuarios.
- **[RF3] Asignación de usuarios en empresas:** Los usuarios con perfil de administrador de la aplicación podrán asignar los usuarios nuevos a cualquier empresa. Los usuarios con rol de administrador de empresa podrán asignar usuarios nuevos a su propia empresa.
- **[RF4] Autenticación:** Los usuarios podrán autenticarse en la aplicación.
- **[RF5] Cerrar sesión:** Los usuarios podrán cerrar sesión en cualquier momento desde la aplicación.
- **[RF6] Modificar contraseña:** Cada usuario podrá modificar su propia contraseña.
- **[RF7] Modificar datos de empresa:** Los usuarios con rol administrador de la empresa podrán modificar los datos de la empresa.

- **[RF8] Eliminación de empresas:** Los usuarios con rol administrador de la aplicación podrán eliminar cualquier empresa eliminando todos los datos asociados a la misma.
- **[RF9] Alta, visualización, modificación y eliminación de proveedores:** Los usuarios con rol administrador de empresa podrán crear, listar, modificar y eliminar proveedores asociados a la empresa.
- **[RF10] Alta, visualización, modificación y eliminación de productos:** Los usuarios podrán crear, listar, modificar y eliminar productos de la empresa a la que están asociados.
- **[RF11] Alta, visualización, modificación y eliminación de categorías:** Los usuarios podrán crear, listar, modificar y eliminar categorías de la empresa a la que están asociados.
- **[RF12] Alta, visualización, modificación y eliminación de ubicaciones:** Los usuarios podrán crear, listar, modificar y eliminar ubicaciones de la empresa a la que están asociados.
- **[RF13] Alta y modificación de compras:** Los usuarios podrán crear y modificar nuevas compras.
- **[RF14] Ubicaciones de compras:** Los usuarios podrán asignar una ubicación origen y destino a una compra.
- **[RF15] Asignar ubicación compra en destino final:** Los usuarios podrán asignar una ubicación de una compra en su destino final.
- **[RF16] Listar compras sin ubicar:** Los usuarios listar aquellas compras que no hayan llegado a su ubicación destino.
- **[RF17] Listar compras filtrado por día:** Los usuarios listar todas las órdenes de compra realizadas en el día seleccionado por el usuario.

### 3.4.2. Requisitos No Funcionales

- **[RNF1] Ejecución de la aplicación:** La aplicación deberá poder ejecutarse desde cualquier sistema operativo móvil.
- **[RNF2] Usabilidad:** Un usuario sin conocimiento ha de poder navegar y utilizar todas las funcionalidades sin problemas.
- **[RNF3] Interfaz:** Los contrastes de los colores cumplirán las pautas de accesibilidad del contenido en la Web 2.0 (WCAG [12]).
- **[RNF4] Carga de aplicación:** El proceso de carga de la aplicación debe ser inferior a 5 segundos.

- **[RNF5] Tiempos de respuesta:** Los tiempos de respuesta ante cualquier consulta deben ser inferiores a 2 segundos.
- **[RNF6] Idioma:** El idioma de la aplicación debe ser en español.
- **[RNF7] Multiusuario:** El servidor debe soportar la conexión de varios usuarios a la vez.
- **[RNF8] Desarrollo:** Las herramientas de desarrollo deben ser de código abierto con licencia gratuita.
- **[RNF9] Test de integración:** En el desarrollo de la aplicación deben ir incluidos los test de integración que garanticen el correcto funcionamiento.
- **[RNF10] Control de acceso a datos:** La información manejada por la aplicación debe estar protegida de accesos no autorizados y de su difusión.
- **[RNF11] Almacenamiento de datos sensibles:** El almacenamiento de datos sensibles como las contraseñas deben estar cifrados.
- **[RNF12] Disponibilidad:** Se debe garantizar una disponibilidad de la aplicación y sus datos del 98 %.
- **[RNF13] Uso de cámara:** La aplicación hará uso de la cámara para la lectura de códigos de barras de los productos.

### 3.5. Modelo Base de Datos

Al utilizar una base de datos no relacional se ha evitado el uso de múltiples conexiones entre los modelos. Como se puede ver en la figura 3.2 toda la aplicación ira orientada en torno a la empresa.

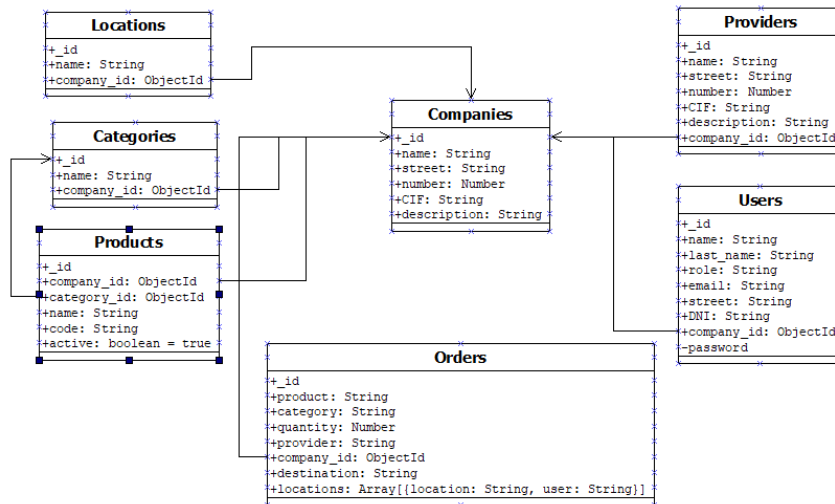


Figura 3.2: Modelo Base de datos.

Todas las entradas se introducirán directamente en texto, sin ids que hacen referencia a otras tablas. Como se ha dicho en capítulos anteriores sobre las bases de datos no relacionales, su fuerte es la velocidad de lectura y escritura de datos no relacionados. Por este motivo se ha decidido tener la funcionalidad que garantice la integridad de los datos en la parte del servidor.

## 3.6. API REST

Para que la aplicación móvil se integre con el servidor se ha desarrollado una API de servicios REST, la cual se encarga de realizar todas las acciones solicitadas por el usuario y devolver el resultado de dicha acción. En el apéndice (A) se pueden ver cada uno de los servicios que ofrece dicha API, en todos los casos la respuesta se da teniendo la sesión iniciada y si no ha sucedido ningún error. En caso de no tener permisos o no tener la sesión iniciada la aplicación devolverá un error 40X con la descripción del error.

## 3.7. Cliente

La filosofía principal de la aplicación es el uso de un diseño fácil e intuitivo, sin necesidad de un aprendizaje complejo ni conocimiento de las últimas tecnologías. Además de su facilidad, lo primero que nos pide el cliente es que sea muy rápido de ver y generar nuevas entradas. Es por esto por lo que se priorizan, viéndose en el inicio y pudiéndose crear desde el mismo. El resto de las gestiones se separan a un segundo plano mediante el menú principal.

### 3.7.1. Mapa de navegación

El mapa de navegación de la aplicación trabaja principalmente desde el menú principal y difiere dependiendo del rol de usuario. Este mapa de navegación está relacionado con los casos de usos tratados en capítulos anteriores. En la figura 3.3 se puede ver el mapa de navegación según el rol del usuario pudiendo este hacer lo mismo que el anterior con sus características añadidas.

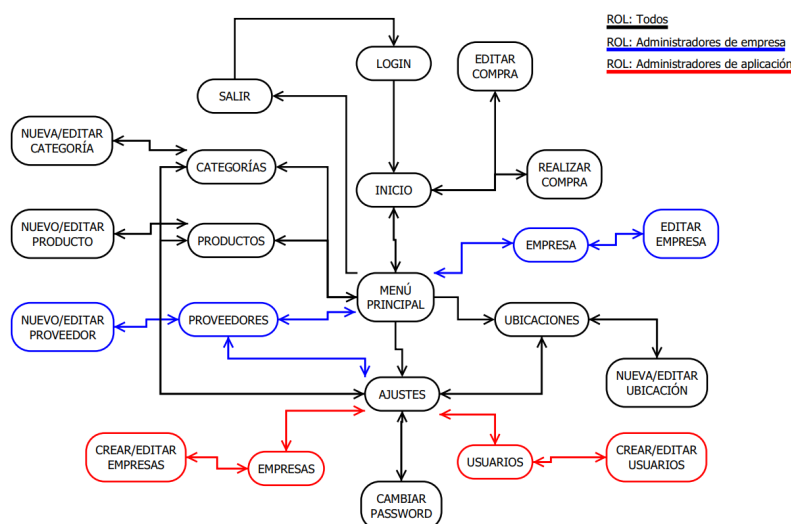


Figura 3.3: Mapa de navegación.

### 3.7.2. Interfaz de usuario

Para la interfaz de usuario se ha separado una pantalla por acción deseada. A continuación, se tratará cada acción por separado:

1. Lo primero que se ve al iniciar la aplicación es el inicio de sesión (figura B.1(a)) si no se tiene una sesión previa. Al introducir usuario y contraseña la aplicación lo enviará al servidor y este responderá con un *token* de sesión o un mensaje de error en caso de que el usuario sea incorrecto o la contraseña sea errónea. Adicionalmente, si el usuario tiene configurado el inicio de sesión por doble paso, el servidor informará de esta configuración a la aplicación y esta le llevará a la pantalla de doble paso (figura B.1(b)) donde pedirá este código y si es correcto le llevará a la pantalla principal (figura B.2). En caso de cerrar la aplicación en mitad del procedimiento el servidor borrará este *token* por estar incompleto y se reiniciará el inicio de sesión pidiendo usuario y contraseña de nuevo (figura B.1(c)).
2. Una vez estamos en el menú principal, podemos crear nuevas entradas. Para ello pulsaremos "nueva entrada". La aplicación nos pedirá seleccionar una categoría



(figura B.3(a)) para filtrar productos (figura B.3(b)). Seleccionado el producto nos solicitará rellenar los datos necesarios para generar la entrada (figura B.3(c)). Ahora podemos ver la entrada creada en el menú principal (figura B.3(d)) si lo giramos a la izquierda, nos aparecerán las posibles acciones que se pueden realizar sobre la entrada B.3(e)). Dichas acciones son: modificar la ubicación, editarla, eliminarla o poner la ubicación como destino final. Las entradas que estén en su destino final aparecerán en azul, el resto en rojo para una mejor identificación visual B.3(f)).

3. Para poder trabajar con un equipo, es necesario crear usuarios con diferentes roles. Para poder crear usuarios uno debe ser administrador de la aplicación o de la empresa y podrá dar roles iguales o inferiores al suyo. Para poder trabajar con estos es tan sencillo como ir al menú de la izquierda (figura B.10(a)) y pulsar usuarios (figura B.4(a)). Aquí se listarán todos los usuarios de la empresa para la que trabaja (en caso de ser administrador de aplicación aparecerán todos los usuarios). Si edita un usuario o lo crea la aplicación le llevará a la página de creación/edición de usuarios (figura B.4(b)), en esta página pedirá obligatorio un nombre, correo, rol y empresa (en caso de ser administrador de empresa solo dejara su propia empresa).
4. El apartado empresas (figura B.5(a)) es solo accesible para usuarios administradores de aplicación desde el menú ajustes (figura B.10(b)). Estos podrán crear/editar todas las empresas de la aplicación. En el caso de los administradores de la empresa, tienen acceso desde el menú lateral a los datos de la propia empresa, donde podrán modificar los mismos.
5. La creación de proveedores se da desde el menú lateral. En él se puede ver la lista de los proveedores que tiene la empresa para la que trabaja (figura B.6(a)), donde se da la posibilidad de crear nuevos (figura B.6(b)). Estos proveedores son los agentes a los que se compran los productos.
6. Para crear productos, primero es necesario crear categorías. La creación de categorías se da en el mismo sitio que su listado (figura B.7), en él se muestran todas las categorías del sistema y si son seleccionables en el momento de crear nuevas compras. Para crear una categoría nueva es tan sencillo como poner el nombre y pulsar a crear. Acto seguido esta categoría aparecerá en la lista como categoría activa. Una vez tenemos las categorías deseadas es el momento de crear los productos (figura B.8), los cuales piden categoría a la que pertenecen y un nombre. Opcionalmente se le puede añadir un código, que en caso de ser código de barras se puede leer con la cámara del móvil y de esta manera se crearan las compras más rápidas sin tener que añadir categoría y producto.
7. Finalmente, para crear ubicaciones (figura B.9) se ha seguido el modelo de categorías, donde se puede crear ubicaciones en el mismo sitio que se listan.
8. Adicionalmente se ha dado la posibilidad de poder modificar la contraseña desde una acción independiente y el poder configurar la seguridad por doble paso. Ambas configuraciones se hacen desde el menú ajustes. Para el cambio de contraseña (figura B.11(a)), la aplicación pedirá la nueva contraseña y la modificará para el usuario que tena la sesión activa. Para la autenticación por doble paso (figura B.11(b)), la

primera vez que se hace, el servidor genera un *token* nuevo y se lo asocia al usuario. El resto de las veces que se active/desactive, el *token* se guarda para posteriores usos. Si se desea modificar el *token* existe la acción cambiar de *token*, el cual dará un *token* nuevo con el que iniciar sesión.

### 3.7.3. Lectura de códigos de barras

Para una gestión más rápida al crear compras se ha añadido códigos de barras en los productos. Se ha aprovechado el *plugin* de Cordova "phonegap-plugin-barcode-scanner" con el que se tiene acceso a la cámara del móvil con lectura de códigos de barras. En la creación de un producto (figura B.8) se ha añadido el campo código de barras con la posibilidad de añadirlo usando la cámara, para esto se ha creado la función *scan* (figura 3.4(a)). En el menú principal se da la posibilidad de escanear un código con el cual se hace una petición al servidor, este devolverá el producto que tiene dicho código o error en caso de que no exista (figura 3.4(b)). Esta característica no funciona sobre exploradores por la ausencia del *plugin* Cordova por lo que se ha necesitado de un móvil Android para probar su correcto funcionamiento.



(a) Añadir contraseña mediante uso de la cámara (b) Búsqueda de producto por código de barras

Figura 3.4: Lectura de códigos de barra.

## 3.8. Pruebas unitarias y TDD

El desarrollo de la aplicación se ha decidido hacer mediante test guiados. Esto supone realizar test que cubran todos los requisitos funcionales antes de implementar dichos requisitos. Esto supone que en un principio todos los test fallarán y según se vayan cubriendo dichos requisitos se irán pasando los test. Para escribir los test, se han utilizado pruebas unitarias que garanticen cada una de las funcionalidades. Estos test se han hecho mediante el uso del *framework* Mocha. Mocha es un *framework* de pruebas para JavaScript

que se ejecutan en Node.js y muestra mediante descripciones que partes del test se pasan correctamente y cuales fallan mediante mensajes descriptivos. En la figura 3.5 se puede ver un ejemplo de test de integración sobre la funcionalidad de poder *loguearse*.



```
it("Should receive a succes true and a token with the authentication", (done) => {
  chai.request(server)
    .post('/login')
    .send({ email: 'pedro@gmail.com', password: 'pedropedro'})
    .end((err, res) => {
      expect(res).to.have.status(200);
      expect(res.body).to.have.property('succes').to.be.equal(true);
      expect(res.body).to.have.property('token');
      expect(res.body).to.have.property('user');
      done();
    })
});
```

Figura 3.5: Test de integración: *Login*.

Estas pruebas se hacen de manera ordenadas, ejecutándose todos los del fichero test. Antes de realizar todas las pruebas se eliminan todos los datos de la BBDD para *test* y se cargan datos nuevos.



## 4 | Principales Vulnerabilidades

En este apartado se tratarán las principales vulnerabilidades que se pueden encontrar en aplicaciones web. Se analizarán sus repercusiones y posibles costes, así como la forma de evitarlas o minimizarlas lo máximo posible. El enfoque adoptado está centrado en la identificación de las principales amenazas de seguridad existentes en el ámbito de las aplicaciones web [1]. En efecto, en el contexto general de la seguridad informática no es posible establecer la seguridad de una solución más que a través de una correcta identificación de las amenazas a principios de seguridad como la confidencialidad, disponibilidad, integridad, autenticación, etc. Identificadas tales amenazas, la ingeniería de la seguridad proporciona un conjunto de procedimientos o salvaguardas orientadas a minimizar el riesgo de que se materialice alguna amenaza y, en caso de la amenaza se concrete, minimizar su impacto [13]. En este trabajo, pues, se estudian las principales amenazas a aplicaciones web basadas en Node.JS y MongoDB. Posteriormente se concreta el análisis en el caso específico de nuestra aplicación web para, en último estadio, ver cómo se tratan los potenciales problemas de seguridad y se trata de impedirlos o reducir al máximo su impacto.

### 4.1. Seguridad en MongoDB

MongoDB es un sistema de bases de datos NoSQL orientado a documentos. En lugar de guardar los datos en tablas fijas, MongoDB se caracteriza por guardar estos datos en documentos con un formato parecido a JSON con estructura dinámica. Esto le da gran potencia en el almacenamiento de datos dinámicos sin tener que definirlos previamente, lo que le otorga una gran flexibilidad a la hora de almacenar datos. Al no tratarse de una base de datos SQL, existe cierta tendencia de pensar que este lenguaje está exento de ataques. En cambio, al igual que en SQL, existen una gran variedad de ataques. Aunque estos dependen más del driver que se utiliza para el tratamiento de datos que de la sintaxis del lenguaje. Dependiendo del lenguaje se puede inyectar código JavaScript en la consulta con el que poder sacar información e incluso ejecutar comandos en el servidor fuera de la base de datos. Siempre que hay un cambio de dominio de trabajo existe el riesgo de una inyección de código, como pasa con la inyección SQL, XPATH o LDAP. En el caso de las bases de datos no relacionales, este cambio de dominio se encuentra desde la entrada de datos del usuario hasta la persistencia de los mismos, es aquí donde el usuario puede introducir código JavaScript que ataque sobre la API JSON que trate los datos NoSQL [14].

En el caso de nuestro proyecto, utilizamos MongoDB con la biblioteca Mongoose. Esta biblioteca evita ciertos ataques JavaScript, pero como se puede ver en la imagen 4.1 no se protege por defecto de una posible inyección de código. En este ejemplo, se ve como se

añade una instrucción “OR” lo que provoca la obtención del primer usuario de la base de datos.

```
function login(req, res) {  
  User.findOne({ email: req.body.email, password: req.body.password }, (err, user) => {  
    if (err) return res.status(500).send({message: `Error durante el login ${err}`});  
    if (!user) return res.status(404).send({message: 'El usuario no existe'});  
    console.log('Usuario logueado:');  
    console.log(user.name)  
    console.log(user.email)  
  })  
}
```

(a) Código consulta MongoDB

```
{  
  "email": { "$ne": " " },  
  "password": { "$ne": " " }  
}
```

(b) Inyección de código

```
Usuario logueado:  
Nuevo  
Nuevo@gmail.com
```

(c) Resultado consulta

Figura 4.1: Ataque NoSQL Injection.

Para protegernos de estos ataques se ha desarrollado el módulo ‘mongo-sanitize’ especializado en limpiar las variables introducidas por el usuario de posibles inyecciones de código malicioso. Para ello se ha utilizado la función `sanitize` (figura 4.2(a)) en cada una de las variables que se reciben en la petición. En la figura 4.2(b), se puede ver el resultado de las variables de la misma consulta realizada en la figura 4.1(b). En cambio, si se envía un texto correcto, la función devuelve el valor de la variable (figura 4.2(c)).

```
function login(req, res) {  
  var email = sanitize(req.body.email)  
  var password = sanitize(req.body.password)  
  
  User.findOne({ email: email, password: password }, (err, user) => {  
    if (err) return res.status(500).send({message: `Error durante el login ${err}`});  
    if (!user) return res.status(404).send({message: 'El usuario no existe'});  
  })  
}
```

(a) Uso de `sanitize` en código

```
Email:  
{}  
Password:  
{}
```

(b) Intento de inyección de código

```
Email:  
Correo valido  
Password:  
Contraseña valida
```

(c) Consulta correcta

Figura 4.2: Protección NoSQL Injection.



A pesar de que se cifren los datos, uno de los ataques más peligrosos con los que se podrían sacar todos los datos de un usuario sin necesidad de atacar a ninguna máquina es el ataque "Man In The Middle"[15, p. 876]. Este ataque consiste en introducirse en la comunicación entre dos equipos y simular ser el equipo contrario (figura 4.5). Al hacerse pasar por la otra máquina la víctima le da sus claves y encripta con las claves del atacante y no con las de la máquina destino final. El interceptor descifra la información, la encripta con la clave real y se la envía al receptor real. De esta manera el atacante es capaz de leer toda la información que se envían cliente y servidor a pesar de que esta esta encriptada. Estos ataques se evitan mediante el establecimiento de un canal seguro. Este canal no es sino un canal cuya confidencialidad está protegida mediante un cifrado simétrico. Ahora bien, para establecer el cifrado en dicho canal es necesario que tanto emisor como receptor tengan la misma clave criptográfica. Aquí nos encontraríamos con el problema de la distribución de claves, que puede ser resuelto mediante criptografía simétrica y una tercera parte confiable como se hace en el caso de Kerberos. No obstante, en un contexto más amplio se hace uso de la criptografía asimétrica para establecer un canal autenticado que, a su vez, permitirá intercambiar una clave de cifrado simétrica o clave de sesión en base a la cual se establece un canal confidencial.

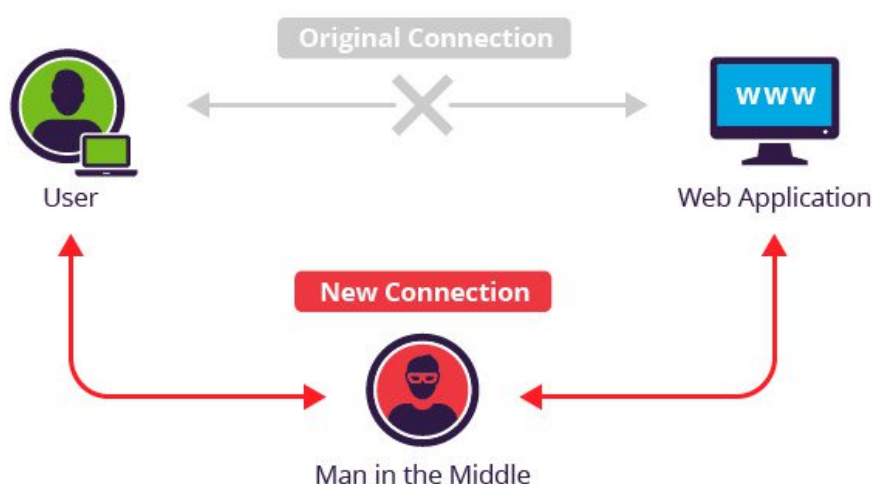


Figura 4.5: Ataque Man in the Middle.

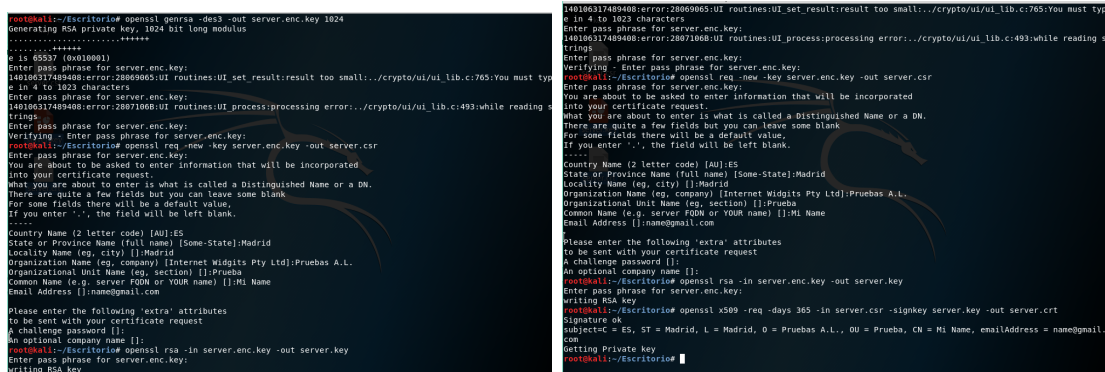
Para corroborar la autoridad de dichas claves públicas, una solución es mediante el uso de autoridades certificadoras. La autenticación mediante autoridades certificadoras consiste en el uso de una tercera entidad confiable que verifique que una clave pública de una entidad es la que dice ser. Esto se hace mediante el uso de clave pública/privada, donde la entidad certificadora obtiene la clave pública que debe verificar mediante un canal seguro, para posteriormente certificar que esta clave pertenece a quien dice ser.

Estos ataques han sido solventados en la aplicación mediante el uso de HTTPS. HTTPS permite establecer un canal autenticado en base al certificado digital del servidor. En ese certificado tenemos la clave pública del servidor firmada por una autoridad de certificación. Dicha clave pública puede emplearse desde el cliente para cifrar una clave de sesión. Solo el servidor puede recuperar esa clave de sesión, ya que solo el servidor



posee la clave privada correspondiente a la clave pública contenida en certificado digital. Recuperada la clave de sesión, el cliente y el servidor intercambian información cifrada empleando la clave de sesión. El problema que existe aquí es la confianza que existe en la autoridad de certificación que firma la clave pública del servidor. Existen múltiples ataques consistentes en la creación de autoridades de certificación falsas. Frente a estos ataques se han creado diversas salvaguardas, siendo la más relevante el estándar Certificate Transparency [9].

Para configurar HTTPS, en la aplicación, primero se han generado las claves publicas/privadas con las que se cifraran las comunicaciones (figura 4.6).



```

root@kali:~/Escritorio# openssl genrsa -des3 -out server.enc.key 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
e=165537 (65537)
Enter pass phrase for server.enc.key:
140186317489408:error:28069065:UI routines:UI_set_result:result too small:../crypto/ui/ui_lib.c:765:You must typ
e in 4 to 1023 characters
Enter pass phrase for server.enc.key:
140186317489408:error:28071068:UI routines:UI_process:processing error:../crypto/ui/ui_lib.c:493:while reading s
trings
Enter pass phrase for server.enc.key:
Verifying - Enter pass phrase for server.enc.key:
root@kali:~/Escritorio# openssl req -new -key server.enc.key -out server.csr
Enter pass phrase for server.enc.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Madrid
Locality Name (eg, city) []:Madrid
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Pruebas A.L.
Organizational Unit Name (eg, section) []:Prueba
Common Name (e.g. server FQDN or YOUR name) []:MI Name
Email Address []:name@gmail.com

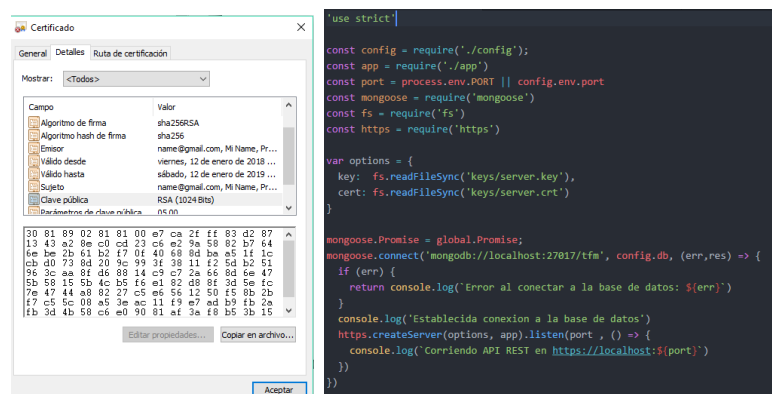
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
root@kali:~/Escritorio# openssl rsa -in server.enc.key -out server.key
Enter pass phrase for server.enc.key:
writing RSA key

root@kali:~/Escritorio# openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
writing RSA key
Signature ok
subject=C = ES, ST = Madrid, L = Madrid, O = Pruebas A.L., OU = Pruebas, CN = MI Name, emailAddress = name@gmail.
com
Getting Private key
root@kali:~/Escritorio#

```

Figura 4.6: Generación claves publicas/privadas.

De aquí se guardarán **server.key** (clave privada del servidor con la que encriptará los mensajes) y **server.crt** (Certificado de la clave pública, en el mismo certificado viene la clave pública con la que desencriptar los mensajes (figura 4.7)). Para leer los certificados se utiliza la biblioteca "fs" con la que leer ficheros y "https" con la que levantar un servidor con conexiones seguras. Para configurar https solo se necesita la clave privada y el certificado de la misma, con esto a partir del comando createServer(claves, aplicación node) se levanta un servicio. En la figura 4.7 se puede ver en código como se inicia un servidor con HTTPS.



(a) Certificado clave pública (b) Levantar servicio node con https

Figura 4.7: Crear un servidor Node.js con conexión HTTPS.

Otro ataque muy popular a los servidores es el de denegación de servicio. Este tipo de ataque no tiene como objetivo la suplantación de identidad ni el robo de información. Si no el conseguir que no pueda responder a ninguna petición consiguiendo que no funcione el servicio para ningún usuario. Lo que puede significar en costes multimillonarios para la empresa al no estar disponibles sus servicios. He incluso que el propio servidor colapse y se dañe físicamente.

Para este tipo de ataque, se ha desarrollado en express un módulo llamado `express-rate-limit`. El cual limita las peticiones posibles que puede realizar un cliente. Ante un ataque de denegación de servicio más complejo utilizando nodos con diferentes IPs se necesitaría una configuración mayor. Pero para el tipo de aplicación, por su popularidad y usuarios finales, con un control en el número de peticiones por IP cumplirá nuestros objetivos. Para configurarlo en el servidor simplemente se necesita importar la librería, configurarla e importarla como *middleware* en todas las peticiones (figuras 4.8).

```
const app = express()
const rateLimit = require('express-rate-limit');

var limiter = new rateLimit({
  windowMs: 15*60*1000, // 15 minutes
  max: 100, // limit each IP to 100 requests per windowMs
  delayMs: 0 // disable delaying - full speed until the max limit is reached
});

app.use(limiter)
```

Figura 4.8: Configuración prevención ataques DDoS.



Figura 4.9: Respuesta ante ataque DDoS.

Con esta sencilla configuración la aplicación rechaza todas las peticiones de una IP si supera las 100, peticiones en 15 minutos (figuras 4.9).

## 4.3. Autenticación

### 4.3.1. Autenticación Básica

#### Sesiones

Una de las principales características de las API REST es su ausencia de estado entre el cliente y el servidor. Al no guardar un estado entre cliente y servidor durante la conexión, la mejor practica para la identificación del cliente es el uso de sesiones.

Una de las técnicas más popularizadas en las API REST para el control de sesiones es el uso de *tokens*. Esta técnica está basada en que, en el momento de autenticarse, el servidor da al cliente un *token* único con el que identifica al usuario. En las siguientes peticiones, el cliente envía este *token* en los datos de la cabecera y el servidor le identifica para poder validar si tiene permisos suficientes para realizar la acción.

JSON Web Token (JWT) es un estándar abierto [16] para generar *tokens* con el fin de enviar datos garantizando de que no han sido modificados y sean seguros. Su uso más común es en servidores web para el inicio de sesiones, donde el servidor envía en un *token* el usuario que ha iniciado sesión y los tiempos de la misma. Después en cada petición el cliente envía el *token* al servidor, este verifica que el *token* no ha sido modificado y la sesión no se ha caducado. La estructura de los JWT se divide en tres partes diferenciadas por puntos, todos excepto la firma codificados en base64 (cabecera.datos.firma):

- **Cabecera:** JSON con información sobre el *token* como el algoritmo de firmado (alg), tipo de *token* (typ), tipo de contenido (alg) etc.
- **Datos:** JSON donde se envían los datos deseados en forma clave: valor.
- **Firma:** Firma utilizando el algoritmo de la cabecera con una clave secreta. Esto garantiza que los datos anteriores no han sido modificados.

Por ejemplo, si queremos enviar el nombre del usuario y si queremos enviar name: "Prueba" y si es admin: true generaríamos el siguiente *token*:

- **Cabecera:**

```
{
  alg: "HS256",
  typ: "JWT"
}
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

- **Datos:**

```
{
  name: "Prueba",
  admin: true
}
eyJhYmV1IjoiUHJ1ZWJhIiwiaWVWRtaW4iOnRydWV9
```

### ■ Firma:

```
SHA256("secret", eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
  ↪ eyJuYW11IjoiUHJ1ZWJhIiwiaWVWRtaW4iOnRydWV9)
= Zc4h4vOJARDB1Li7U0XsobxaDofH-3GcyuCwyV8gM2g
```

```
JWT: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
  ↪ eyJuYW11IjoiUHJ1ZWJhIiwiaWVWRtaW4iOnRydWV9.
  ↪ Zc4h4vOJARDB1Li7U0XsobxaDofH-3GcyuCwyV8gM2g
```

A primera vista se puede ver como gracias al uso de una clave única para firmar garantiza que estos datos no han sido modificados ni generados por terceras partes. Por otro lado, tenemos el inconveniente de que los datos que se envían son fácilmente descifrables, pudiendo identificar el id del usuario o su rol. Además, en caso de ser robado el *token* no podrá ser censurado con relativa facilidad dando acceso por tiempo ilimitado o por un tiempo en caso de poner fecha de expedición en el *token*. Una forma para evitar esto es guardar toda la información del *token* en una base de datos y enviar simplemente el id de la tabla. En caso de robo de sesión se podrá eliminar fácilmente borrando la sesión de la tabla, de esta manera cada vez que le llegue el *token* borrado lo buscará en la base de datos y no encontrará la sesión referida rechazando la conexión.

En el caso de la aplicación se ha creado una función de creación de sesión y una exclusiva para la validación de los *tokens* de sesión (figura 4.10). En el primer caso devuelve el *token* de sesión, mientras que en el segundo caso devuelve el usuario *logueado* con ese *token*.

## Control de acceso

De poco sirve el asegurar una aplicación si no se tiene un control de acceso sobre ciertos lugares. Muchas páginas te redirigen a una URL u otra dependiendo del rol que tengas (administración/usuario) y dentro de la página solo dan acceso a ciertas URLs consiguiendo así que un usuario común no entre en la parte de administración de este sistema. El problema viene cuando el usuario introduce la URL a la que no tiene permiso directamente y accede a la misma. Si la aplicación no tiene un control de acceso y no le deniega la petición en caso de no tener permisos, el usuario puede realizar las acciones de administrador sin control alguno. Esta vulnerabilidad es fácil de eliminar mediante un control de acceso por *middleware* que garantice el inicio de sesión y rol del usuario antes de responder cualquier petición.

```
function createToken (user) {
  const token = new Promise((resolve, reject) => {
    try {
      const session = new Session()
      const moment = new Date()
      session.user = user._id
      session.iat = moment
      session.exp = new Date()
      session.exp.setDate(session.iat.getDate() + config.tokens.exp);

      session.save((err, session) => {
        if (err) reject({ status: 401, message: 'Error al generar la sesion ${err.errmsg}' })

        const payload = {
          sub: session._id,
          iat: session.iat,
          exp: session.exp
        }

        resolve(jwt.encode(payload, config.tokens.SECRET_TOKEN))
      })
    } catch (err) {
      reject({
        status: 500,
        message: 'Invalid token'
      })
    }
  })
  return token
}

function decodeToken (token) {
  const decoded = new Promise((resolve, reject) => {
    try {
      const payload = jwt.decode(token, config.tokens.SECRET_TOKEN)
      const moment = new Date()
      if (payload.exp <= moment){
        reject({
          status: 401,
          message: 'El token ha expirado'
        })
      }
      Session.findById(payload.sub, (err, session) => {
        if (err) reject ({ status: 404, message: 'Error al buscar la session' });
        if (!session) reject ({ status: 404, message: 'La session ha sido eliminada' });
        resolve(session.user)
      })
    } catch (err) {
      reject({
        status: 404,
        message: 'La session ha sido eliminada'
      })
    }
  })
  return decoded
}
```

(a) Creación de una sesión nueva

(b) Validación de un *token* recibido

```
{
  "_id": ObjectId("5a95da901ba9de0b30d3bc6c"),
  "exp": 2018-03-13 23:24:16.884,
  "iat": 2018-02-27 23:24:16.883,
  "user": "5a4d4e2c15351631bc7079e1",
  "__v": 0
}
```

(c) *Token* guardado en la base de datosFigura 4.10: Creación y validación de *tokens* de sesión.

En primer lugar, hay que crear un *middleware* que garantice que hay un usuario *logueado* en todas las rutas excepto la de *login* (figura 4.11).

```
// Routes
// **Login
app.post('/login', authCtrl.login)

// Routes auth
app.use("/app", authMiddleware)
app.use("/app", routes)
```

Figura 4.11: *Middleware Login*.

Una vez se ha garantizado que el usuario ha iniciado sesión hay que realizar un control de roles de acceso y acciones. Para el control de roles se han creado tres *middlewares* (figura 4.12) que cubren las posibles heurísticas de la aplicación:

- **adminFilter:** Si el usuario no es súper administrador se añade filtro de compañía para que no pueda acceder a datos que no sean de su compañía.
- **superAdminAction:** Solo el rol de súper administrador puede realizar esta acción, para el resto de los usuarios la petición es rechazada.

- **companyFilter**: Solo se puede realizar esta acción si la compañía sobre la que se realiza la acción es la misma a la que pertenece el usuario. Esto evita el poder modificar datos de una compañía a la que no pertenece el usuario.

```
function adminFilter(req, res, next) {
  if(req.user.role != 'SuperAdmin'){
    req.filter = {company_id: req.user.company_id}
  }else{
    req.filter = {}
  }
  next()
}

function superAdminAction(req, res, next) {
  if(req.user.role != 'SuperAdmin'){
    return res.status(404).send({message: 'No tiene privilegios suficientes'});
  }else{
    next()
  }
}

function companyFilter(req, res, next) {
  if(req.user.role != 'SuperAdmin'){
    if(req.params.id && req.user.company_id != req.params.id){
      return res.status(404).send({message: 'No tiene privilegios suficientes para esta empresa'});
    }
    req.filter = {id: req.user.company_id}
  }else{
    req.filter = {}
  }
  next()
}
```

```
// **Companies
router.get('/company/:id', filterMiddleware.companyFilter, companyCtrl.getCompany)
router.get('/companies', filterMiddleware.companyFilter, companyCtrl.getCompanies)
router.post('/company', filterMiddleware.superAdminAction, companyCtrl.newCompany)
router.put('/company/:id', filterMiddleware.companyFilter, companyCtrl.updateCompany)
router.delete('/company/:id', filterMiddleware.superAdminAction, companyCtrl.deleteCompany)

// **Providers
router.get('/provider/:id', filterMiddleware.adminFilter, providerCtrl.getProvider)
router.get('/providers', filterMiddleware.adminFilter, providerCtrl.getProviders)
router.post('/provider', providerCtrl.newProvider)
router.put('/provider/:id', filterMiddleware.adminFilter, providerCtrl.updateProvider)
router.delete('/provider/:id', filterMiddleware.adminFilter, providerCtrl.deleteProvider)
```

(a) Control de acciones

(b) Uso de *middleware* en rutas específicasFigura 4.12: Control de acceso por *middlewares*.

### 4.3.2. Autenticación Avanzada

#### Verificación en dos pasos

Para mejorar la seguridad en el inicio de sesión se ha añadido la posibilidad iniciar sesión mediante doble paso. Node.js no incluye un mecanismo propio, se ha necesitado la librería "speakeasy".

A partida de la función `generateSecret()` creamos un código único que el usuario debe guardar en un lugar seguro. En nuestro caso, lo guardaremos en la aplicación "Google Authenticator". Este código secreto genera una clave aleatoria cada 30 segundos. Esta clave será solicitada en el inicio de sesión y verificada mediante la función `"speakeasy.time.verify(secret: secret.base32, encoding: 'base32', token: token)"`. En caso de que sea correcta se añadirá un *flag* de paso en el *token* con el que podrá tener acceso al resto de la aplicación. Si se hace una petición con un *token* de sesión que no tiene el *flag* activo será rechazada la petición y eliminada la sesión ya que se considera fraudulenta. Se ha seguido este mecanismo siguiendo las recomendaciones de NIST, evitando de esta forma el uso de SMS como canal seguro de transmisión de la contraseña de corta vida[17].

## 4.4. Exposición de datos sensibles

Muchas aplicaciones no protegen debidamente datos sensibles de los usuarios como pueden ser contraseñas o cuentas bancarias. Esto supone que, ante un robo de información, estos datos quedan expuestos para el atacante.

Para protegerse de esta vulnerabilidad, lo primero que hay que hacer es cifrar todos los datos que se envían. Para ello se ha usado SSL con lo que se evita *man in the middle* y se cifran los datos que se envían, este tema ya se ha tratado en temas anteriores por lo que no lo repetiremos. Los principales ataques no intentan romper un cifrado debido a su complejidad, por lo que en vez de sacar datos de los mensajes, muchos ataques van sobre los datos almacenados en el servidor.

Para evitar el robo de datos sensibles, en primer lugar no se deben guardar datos sensibles que no sean necesarios como pueden ser datos de salud o bancarios. Para los datos que se tienen que guardar y que puedan ser sensibles en caso de robo, se deben utilizar algoritmos de encriptación seguros que garanticen que estos datos no pueden ser sacados en caso de robo. En el caso de este proyecto, el dato más sensible es el de las contraseñas de los usuarios. Este será protegido con el algoritmo de *hash* bcrypt [18]. Para guardarlo cifrado, Mongoose proporciona un *middleware* que permite modificar los datos de un modelo antes de guardarlo en la base de datos. En la figura 4.14(a), se puede ver como se genera un hash a partir de la contraseña recibida. Esto genera un código único e indescifrable, aun teniendo la semilla con la que se ha generado el *hash*. Bcrypt es un algoritmo basado en Blowfish, al cual le añade *salt* (un fragmento aleatorio para generar el *hash*) evitando que un *hash* asociado a una contraseña sea único, este *salt* se genera de forma aleatoria cada vez que se cifra una cadena por lo que dos contraseñas iguales (figura 4.13) obtienen dos *hashes* diferentes. Para comparar una nueva contraseña con la guardada Bcrypt utiliza el *salt* del cifrado y compara el *hash* resultante (figura 4.14(b)).

```

Password:
Password
Salt:
$2a$10$79tVl0pII2GcnobbRpxvge
Bcrypt:
$2a$10$79tVl0pII2GcnobbRpxvgep5p4iN6zIb1LwBy5zJDw4bnV7Zyd0b6

```

(a) Primer cifrado cadena 'Password'

```

Password:
Password
Salt:
$2a$10$BXgYYW.AMZ0v8wMUbh1jS.
Bcrypt:
$2a$10$BXgYYW.AMZ0v8wMUbh1jS.NRR3YTQ6w09j6f0XhtDkek1tHJi.R5.

```

(b) Segundo cifrado cadena 'Password'

Figura 4.13: Cifrado de contraseñas con bcrypt.

Este algoritmo es más costoso que otros conocidos como MD5, SHA1, etc. Pero la

generación de texto aleatorio supone una protección extra frente ataques como tablas arcoíris o tablas de consultas que tienen guardados los resultados a ciertas palabras facilitando un ataque por fuerza bruta.

```
UserSchema.pre('save', function (next) {
  let user = this
  if(!user.isModified('password')) return next()

  bcrypt.genSalt(10, (err, salt) => {
    if (err) return next(err)

    bcrypt.hash(user.password, salt, null, (err, hash) => {
      if (err) return next(err)
      user.password = hash
      next()
    })
  })
})
```

(a) Cifrado de contraseña antes de guardarla

```
User.findOne({ email: req.body.email }, (err, user) => {
  if (err) return res.status(500).send({message: `Error durante el login ${err}`});
  if (!user) return res.status(404).send({message: 'El usuario no existe'});

  bcrypt.compare(req.body.password, user.password, function(err, resultado) {
```

(b) Comparación de la contraseña del usuario con la recibida

Figura 4.14: Protección de datos sensibles.

## 4.5. XXE

XXE (Xml eXterna Entity) es una vulnerabilidad que se produce en las aplicaciones que hacen uso de intérpretes de XML. Aplicaciones que reciben como entrada un documento XML y realizan un *parseo* sin control del mismo. El atacante puede aprovecharse de esta vulnerabilidad enviando un XML formateado para que el intérprete envíe información del sistema, ejecute comandos o consigan una denegación del propio servicio.

Estos ataques son posibles cuando el intérprete de XML permite incluir entidades externas, introduciendo etiquetas propias en el XML. Una entidad, es una forma sencilla de crear un alias, para reducir así líneas de código. Estos alias son llamados dentro del XML mediante el carácter `&nombre_alias`. Por ejemplo, si se quiere escribir en varios sitios el nombre de un autor se añade `<!DOCTYPE autor [<!ENTITY terminos ‘Nombre del autor’>]>` al inicio del documento y cada vez que se quiere que aparezca se pone `&autor`, de esta manera se evita



que si hay que cambiarlo, se tenga que cambiar en muchos sitios. El peligro de estas etiquetas es que pueden llamar a cualquier binario como puede ser lectura de archivos e incluidos hacer denegaciones de servicio.

Para evitar este tipo de ataques, se tiene que configurar el intérprete XXE para que no ejecute entidades externas y evitar así posible código malicioso. Actualmente, en la aplicación no aplica ya que no hay subida de datos mediante XML, pero se ha nombrado ya que en un trabajo futuro se plantea la subida masiva de datos como pueden ser los datos de empresas, productos, etc.

## 4.6. XSS

XSS (Cross site scripting) es una de las 10 vulnerabilidades más encontradas en las aplicaciones web [19]. Esta vulnerabilidad consiste en la infección de código como parte del HTML que envía el navegador sin que realmente sea de él. Este tipo de ataque comenzó en la década de los 90 sin un nombre que lo defina hasta el año 2000, cuando Microsoft le puso nombre. Empresas como Facebook, Twitter han sufrido este tipo de ataque en sus aplicaciones. Existen dos tipos de ataques XSS:

- **Reflejado:** Es el ataque XSS más habitual, consiste en inyectar el código atacante por la URL. Mediante variables que posteriormente va a leer el intérprete. Al acceder un usuario mediante la URL infectada, su navegador ejecutara código malicioso al interpretar los valores de la URL. Este ataque depende directamente la URL que se envía a la víctima. El resto de los usuarios no sufren el ataque. Por ejemplo, hay páginas que envían el nombre del usuario por la URL y la página lo recoge para saludar con el nombre. Si en campo el nombre se inyecta código HTML, este código se mostraría en la página como parte de la misma. Igual pasaría con JavaScript o cualquier lenguaje que pueda interpretar el navegador pudiendo robar sesiones etc.
- **Persistente:** Este tipo de ataque consiste en embeber código malicioso directamente sobre la página web mediante el uso de formularios etc. Es el más peligroso de los dos tipos ya que este ataque se ve reflejado sobre todos los usuarios que accedan a la web. Dentro de este grupo existe un subgrupo que consiste en atacar directamente sobre el DOM (Document Object Model) mal configurado consiguiendo que se abran otras páginas no deseadas en la web atacada. Como ejemplo sencillo, en un foro se puede insertar código en los comentarios, este código se guarda en la BBDD y lo ejecuta en todos los usuarios que se metan en el post.

Estos ataques pasan por un exceso de confianza por parte del cliente sobre los datos que recibe para mostrar. Para proteger de estos ataques existen el uso de librerías que *parsean* los datos antes de mostrarlos. En el caso del *framework* utilizado, viene configurado de forma nativa para estar protegido de estos ataques, en la figura 4.16 se ve como los caracteres `<` y `>` se pasan a `\&lt;` y `\&gt;`, consiguiendo que el intérprete lo muestre bien no considerándolo como parte del código. En la figura 4.15 vemos como se imprime en formato texto el código HTML que se ha intentado inyectar.



Figura 4.15: Protección XSS Ionic.

```
<ion-label class="label label-md" id="lbl-17">&lt;/ion-label&gt; &lt;button (click)="alert(1)"&gt;&lt;/button&gt;&lt;ion-label&gt;"Prueba"</ion-label>
```

Figura 4.16: Código parseado XSS.

## 4.7. CSRF

La técnica de falsificación de petición en sitios cruzados, Cross Site Request Forgery (CSRF o XSRF) consiste en conseguir enviar de forma no voluntaria una petición sobre un servicio validado en el navegador. Esta petición será procesada en nombre del usuario pasando los controles de seguridad de *roles/login*. Estos ataques son debidos al exceso de confianza por parte del servidor que cree que todas las peticiones que reciben son ilícitas.

Para comprender este ataque pondremos un ejemplo que podemos seguir con la figura 4.17:

Un usuario se *loguea* en una página web. Al tiempo decide abrir una ventana y acceder a la página atacante. Esta página atacante hace una petición POST sobre la página web en la que estaba *logueado* el usuario pidiendo un cambio de contraseña. El servidor víctima mira el usuario *logueado* por sesión y le cambia la contraseña. El atacante pasa a tener acceso a la web mediante la contraseña modificada.

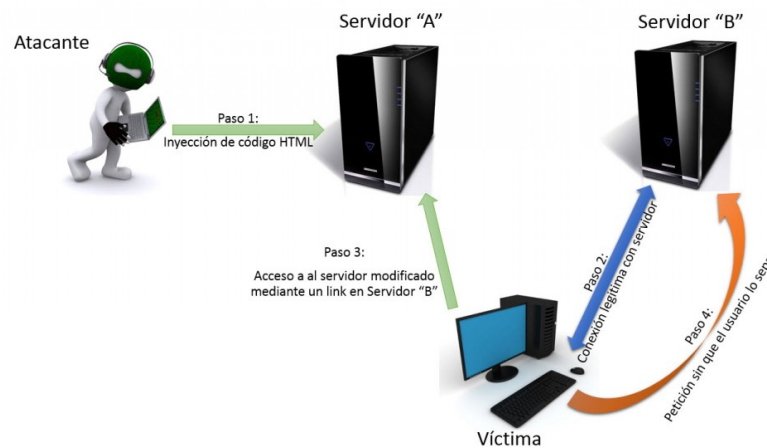


Figura 4.17: Esquema Ataque CSRF.

En la aplicación este tipo de ataque no aplica debido a las siguientes razones:

- **No se trabaja directamente sobre el servidor:** Ionic no realiza peticiones post sobre el formulario, si no sobre métodos. El navegador guarda los datos de cabecera sobre la dirección IP del cliente, no del servidor por lo que no se pueden copiar los datos.
- **No se usan cookies:** En el caso de esta aplicación se ha decidido el no utilizar *cookies* de sesión, lo que evita su posible robo.

## 4.8. Configuraciones Incorrectas

Muchas grietas de seguridad se encuentran en la ausencia de configuración de las aplicaciones al instalarlas y usarlas. Muchas páginas web y bases de datos tienen la configuración por defecto que tiene durante la instalación. Estas configuraciones tienen como objetivo el correcto funcionamiento de la aplicación y acceso a la misma desde cualquier punto sin hacer controles de roles, intentos de acceso, etc. Esto facilita el poder atacar al *framework* utilizado desde cualquier localización, e incluso encontrar por internet la contraseña por defecto que tiene el rol administrador en la aplicación dando un control total al atacante.

En el caso de MongoDB, no existe un control de usuario por defecto dando acceso a todos los usuarios que lo deseen. Esto se debe a que la filosofía principal de MongoDB no es el control de roles de acceso, si no la disponibilidad y velocidad de obtención de datos. De todas maneras, se puede configurar un control de roles de acceso sobre las tablas. Para esto hay que crear un fichero de configuración que limite dicho acceso y los usuarios con sus roles correspondientes (figura 4.18).

Creado el fichero de configuración y dados de alta los usuarios con sus roles, hay que arrancar MongoDB con el *flag* `--config` fichero configuración para que lea dicho

(a) Fichero Configuración MongoDB (b) Creación de usuarios en MongoDB

(a) Consultas sin permisos

(b) Consultas con permisos

alguien ha intentado acceder a los datos y que consultas se han realizado.

Para Node.js, lo que hay que configurar son todos los módulos utilizados. Estos módulos suelen estar configurados para que puedan ser utilizadas en cualquier circunstancia por lo que tienden a tener todos los accesos abiertos y configurados para poder realizar cualquier acción sin restricción.

A continuación, listaremos los módulos más importantes y su correspondiente configuración:

- **body-parser:** De la librería *body-parse*, solo se *parsean* los datos *.json*. Por lo que se puede configurar que se ignoren el resto de los tipos de datos. En la configuración se puede modificar el tipo de dato a analizar, tamaño máximo, aceptación de datos comprimidos etc.
- **cors:** El objetivo de utilizar CORS es el aceptar peticiones de orígenes distintos a los del propio servidor mediante propiedades en el *header*. Al utilizar un cliente ajeno tenemos que dar permiso a todos los orígenes ya que se puede acceder desde cualquier móvil, no desde un origen fijo. También es aquí donde se configuran las peticiones y cabeceras aceptadas.
- **helmet:** Helmet ayuda a proteger los servicios web mediante el uso correcto de cabeceras. Establece que datos se envían y cuáles no, consiguiendo de esta manera que el atacante no tenga un exceso de información que le ayude a centrarse en el objetivo.

## 4.9. Versiones Antiguas

Con el despliegue de nuevas versiones no solo se dan nuevas características, si no que se cubren vulnerabilidades encontradas en el propio *framework*/lenguaje. Antes de actualizar hay que garantizarse que no se pierden características y el programa sigue funcionando correctamente, pero no excusa para no actualizar ya que se publican a diario que vulnerabilidades tiene cada versión y debido a esto es más fácil para el atacante el atacar sobre un software desactualizado que contra uno que está en la última versión.

Lo mejor para este tipo de vulnerabilidad es estar en la lista de correo de vulnerabilidades de los *frameworks* utilizados y mirar en cada actualización cuales son las características nuevas y que errores cubre. Conociendo las vulnerabilidades del propio sistema valorar los daños que se pueden sufrir y proceder a invertir en tantos recursos se necesite como de grave sea la vulnerabilidad. Si vamos a la página oficial de "Express Security", tienen un apartado dedicado exclusivamente para mostrar que vulnerabilidades tienen cada versión del módulo.

Además de esto, existe un proyecto en Node.js llamado 'Node Security Project', el cual analiza todas las librerías del proyecto y chequea la versión que se utiliza con una base de datos de vulnerabilidades mostrando el estado de las mismas. Esta librería se

instala a nivel global en la máquina de desarrollo, no a nivel de aplicación por lo que se debe instalar mediante el comando `npm -g install nsp`. Para realizar el test solo hay que ejecutar el comando `nsp check --output summary` e imprimirá en la terminal el resultado (figura 4.20).

Regular Expression Denial of Service	
Name	mime
CVSS	7.5 (High)
Installed	1.3.4
Vulnerable	< 1.4.1    > 2.0.0 < 2.0.3
Patched	>= 1.4.1 < 2.0.0    >= 2.0.3
Path	server@1.0.0 > express@4.15.4 > send@0.15.4 > mime@1.3.4
More Info	<a href="https://nodesecurity.io/advisories/535">https://nodesecurity.io/advisories/535</a>
Regular Expression Denial of Service	
Name	debug
CVSS	3.7 (Low)
Installed	2.6.8
Vulnerable	<= 2.6.8    >= 3.0.0 <= 3.0.1
Patched	>= 2.6.9 < 3.0.0    >= 3.1.0
Path	server@1.0.0 > body-parser@1.18.1 > debug@2.6.8
More Info	<a href="https://nodesecurity.io/advisories/534">https://nodesecurity.io/advisories/534</a>
Regular Expression Denial of Service	
Name	fresh
CVSS	7.5 (High)
Installed	0.5.0
Vulnerable	< 0.5.2
Patched	>= 0.5.2
Path	server@1.0.0 > express@4.15.4 > fresh@0.5.0
More Info	<a href="https://nodesecurity.io/advisories/526">https://nodesecurity.io/advisories/526</a>
Regular Expression Denial of Service	
Name	debug
CVSS	3.7 (Low)
Installed	2.6.8
Vulnerable	<= 2.6.8    >= 3.0.0 <= 3.0.1
Patched	>= 2.6.9 < 3.0.0    >= 3.1.0
Path	server@1.0.0 > express@4.15.4 > debug@2.6.8
More Info	<a href="https://nodesecurity.io/advisories/534">https://nodesecurity.io/advisories/534</a>

Figura 4.20: Vulnerabilidades en versiones antiguas.

Con este método se garantizan en gran medida el no tener versiones vulnerables. Ya que como se puede ver, no solo se depende de los módulos instalados, si no de las dependencias a otros módulos que tienen estos. En el caso de este proyecto la versión de express 4.15.4 dependía de los módulos debug (2.6.8), fresh (0.5.0) y mime (1.3.4) con vulnerabilidades de alto riesgo en dos de los tres casos. Actualizando todos los módulos a la última versión se solucionan estas vulnerabilidades (figura 4.21).

```

PS D:\Users\Pedro\Desktop\TFM_Pedro\api-server> npm update
npm WARN server@1.0.0 No repository field.

+ express@4.16.2
+ mongoose@4.13.11
+ body-parser@1.18.2
+ nodemon@1.17.1
added 7 packages, removed 6 packages and updated 31 packages in 12.031s
PS D:\Users\Pedro\Desktop\TFM_Pedro\api-server> nsp check --output summary
(+ ) No known vulnerabilities found
  
```

Figura 4.21: Actualización a la última versión.

En caso de estar en producción, cada vulnerabilidad provee de un enlace con más información sobre la misma. Con esto se debe hacer una valoración de tiempo de test para ver que la nueva versión no provoca errores en la aplicación y la necesidad por subir a la nueva versión.

## 5 | Conclusiones y Trabajo Futuro

### 5.1. Conclusiones

A lo largo del presente trabajo se ha realizado el desarrollo de una aplicación cliente/servidor con el uso de la tecnología JavaScript con los *frameworks* Node.js e Ionic para posteriormente hacer un estudio sobre la seguridad de la misma. Cabe destacar que éste trabajo ha conllevado la adquisición de numerosos conocimientos no adquiridos durante el máster, así como el refuerzo de ciertos conceptos que se habían tratado a lo largo del mismo, llevándolos a un nuevo nivel con su puesta en práctica en entornos reales.

El aspecto principal de este proyecto ha sido el estudio sobre la seguridad que rodean las tecnologías web. Para ello se ha decidido el uso de la tecnología Node.js debido al gran potencial del mismo, a su popularidad y uso en el mercado real. Para bases de datos se ha utilizado modelos no relacionales, en concreto MongoDB. El objetivo de la aplicación es la acumulación de datos y localizaciones sin una complejidad de relación alta. Debido a su gran potencial con el tratamiento de datos se ha decantado por esta tecnología. Para el tema del cliente, al ser una tecnología que tiene que ser ejecutada en diferentes dispositivos con S.O. diferentes se ha decantado por el uso de tecnologías híbridas. En nuestro caso Ionic, debido a su parecido a Angular y el uso del lenguaje JavaScript. Gracias a esto se ha estudiado el desarrollo de aplicaciones web mediante un diseño centralizado en el lado del cliente y uso de Node.js. El uso de estas tecnologías ha permitido el despliegue el mismo en sistemas *cloud*, evitando de esta manera la dependencia de contar con un servidor web consiguiendo el objetivo de ubicuidad en lo relacionado a *backend*. El carácter agnóstico de la aplicación en lo relativo al servidor se potencia en el lado del cliente habilitando el uso no sólo desde navegadores web, sino también a través de dispositivos móviles mediante el desarrollo de una aplicación SPA (Single Page Application) haciendo uso de Ionic.

En cuanto al estudio de la seguridad se han analizado las principales vulnerabilidades web, para esto se han analizado el top 10 de vulnerabilidades web según el proyecto OWASP (Open Web Application Security Project) [7], un proyecto de código abierto dedicado a descubrir y combatir las principales inseguridades de las aplicaciones web. Durante este estudio se ha visto como las principales inseguridades no dependen de la tecnología, si no del exceso de confianza y la falta de configuraciones.

Las principales vulnerabilidades encontradas son la falta de configuraciones por parte de las herramientas a utilizar, el envío de datos sin cifrar dando acceso a toda la información desde el exterior. Y, sobre todo, al exceso de confianza. El exceso de confianza puede venir desde cualquier sitio, ya sea desde los datos introducidos por el usuario, como pueden ser, SQL injection, XSS o XXE. Exceso de confianza por parte del servidor al confiar plenamente en el cliente como se pueden ver ataques CSRF o exceso de confianza

por parte del cliente sobre el servidor como se dan en los ataques XSS persistentes.

Para evitar vulnerabilidades propias de las tecnologías utilizadas, es necesario tenerlas lo más actualizadas posible. Ya que no solo dan mejoras de funcionalidad o rendimiento, si no que cubren vulnerabilidades descubiertas. Las herramientas más utilizadas suelen tener un apartado propio de seguridad, donde exponen las vulnerabilidades que sufren cada versión de la herramienta.

Este proyecto ha servido como ampliación de las asignaturas Internet y Redes Avanzadas (IRAV) y Seguridad y Auditoría de los Sistemas de Información (SASI), permitiendo mejorar mi perfil como desarrollar *full-stack* que implementa y diseña sus aplicaciones de acuerdo con los principios de *security-by-design* [20].

### 5.2. Trabajo Futuro

Los pasos siguientes a este proyecto será trabajar en tener un sistema de registro de incidencias que permita la identificación efectiva de problemas de seguridad en la aplicación. En efecto, el actual diseño incluye los objetivos de Confidencialidad, Integridad, Disponibilidad (esto último derivado de la utilización de servicios *cloud* que proporcionan acceso a los recursos en cualquier instante), Autenticación y Autorización, a falta de una mejor política de monitorización de actividad para tener el requisito de Auditabilidad (que se basa en buenos *logs*, accesibles únicamente a los administradores de sistemas y a las audiciones de seguridad). Estas tres últimas reglas son denominadas las reglas de oro de la seguridad. En relación con la autenticación y los logs, también se estimarán opciones que permitan efectuar registros de actividad en modo anónimo, esto es, sin que esto suponga un perjuicio a la privacidad de los usuarios y teniendo en mente la compatibilidad de la aplicación con la nueva normativa europea de protección de datos privados (GDPR).

Como mejora para la creación de usuarios, se propone el incluir un verbo en la *API REST* que efectúe el registro de usuarios e incluir dicha funcionalidad en el lado del cliente. Este sistema de registro debe seguir los protocolos necesarios que verifiquen los nuevos registros [21].



## A | API REST

/login	
Método	POST
Descripción	
El servidor comprueba si el usuario existe en la base de datos. Una vez se obtiene el usuario, se valida si la contraseña introducida es la misma que la contraseña del usuario.	
Datos entrada	<ul style="list-style-type: none"><li>▪ <b>username:</b> nombre de usuario.</li><li>▪ <b>password:</b> Contraseña del usuario.</li></ul>
Respuesta	<ul style="list-style-type: none"><li>▪ <b>Usuario y contraseña correcto:</b><ul style="list-style-type: none"><li>• <b>Código:</b> 200</li><li>• <b>Respuesta:</b><ul style="list-style-type: none"><li>◦ <b>user:</b> Datos del usuario</li><li>◦ <b>success:</b> true</li><li>◦ <b>token:</b> token de sesión</li></ul></li></ul></li><li>▪ <b>Usuario o contraseña incorrecto:</b><ul style="list-style-type: none"><li>• <b>Código:</b> 404</li><li>• <b>Respuesta:</b><ul style="list-style-type: none"><li>◦ <b>message:</b> Motivo del error</li><li>◦ <b>success:</b> false</li></ul></li></ul></li></ul>

Tabla A.1: API REST: POST /login.

/app/companies	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve todas las compañías del sistema	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>companies:</b> Array con las compañías del sistema</li> </ul> </li> </ul>

Tabla A.2: API REST: GET /app/companies.

/app/company/:companyId	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve la compañía cuyo id es igual a :companyId	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>company:</b> Datos de la compañía</li> </ul> </li> </ul>

Tabla A.3: API REST: GET /app/company/:companyId.

/app/company	
<b>Método</b>	<b>POST</b>
<b>Descripción</b>	
El servidor crea una nueva compañía con los datos recibidos	
<b>Datos entrada</b>	<ul style="list-style-type: none"> <li>▪ <b>company:</b> Datos de la compañía.</li> </ul>
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>company:</b> Datos de la compañía guardada</li> </ul> </li> </ul>

Tabla A.4: API REST: POST /app/company.

/app/company/:companyId	
<b>Método</b>	<b>PUT</b>
<b>Descripción</b>	
El servidor modifica con los datos recibidos la compañía cuyo id es igual a :companyId	
<b>Datos entrada</b>	<ul style="list-style-type: none"> <li>▪ <b>company:</b> Datos a modificar de la compañía.</li> </ul>
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>company:</b> compañía actualizada</li> </ul> </li> </ul>

Tabla A.5: API REST: PUT /app/company/:companyId.

/app/company/:companyId	
<b>Método</b>	<b>DELETE</b>
<b>Descripción</b>	
El servidor elimina la compañía cuyo id es igual a :companyId	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> </ul>

Tabla A.6: API REST: DELETE /app/company/:companyId.

/app/providers	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve todos los proveedores del sistema/empresa dependiendo del rol del usuario	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>providers:</b> Array con los proveedores resultantes.</li> </ul> </li> </ul>

Tabla A.7: API REST: GET /app/providers.

/app/provider/:providerId	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve el proveedor cuyo id es igual a :providerId	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>provider:</b> Datos del proveedor</li> </ul> </li> </ul>

Tabla A.8: API REST: GET /app/provider/:providerId.

/app/provider	
<b>Método</b>	<b>POST</b>
<b>Descripción</b>	
El servidor crea un nuevo proveedor con los datos recibidos	
<b>Datos entrada</b>	<ul style="list-style-type: none"> <li>▪ <b>provider:</b> Datos del proveedor.</li> </ul>
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>provider:</b> Datos del proveedor guardada</li> </ul> </li> </ul>

Tabla A.9: API REST: POST /app/provider.

/app/provider/:providerId	
Método	PUT
Descripción	
El servidor modifica con los datos recibidos del proveedor cuyo id es igual a :providerId	
Datos entrada	<ul style="list-style-type: none"> <li>▪ <b>provider:</b> Datos a modificar del proveedor.</li> </ul>
Respuesta	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>provider:</b> proveedor actualizado</li> </ul> </li> </ul>

Tabla A.10: API REST: PUT /app/provider/:providerId.

/app/provider/:providerId	
Método	DELETE
Descripción	
El servidor elimina el proveedor cuyo id es igual a :providerId	
Datos entrada	
Respuesta	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> </ul>

Tabla A.11: API REST: DELETE /app/provider/:providerId.

/app/categories	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve todas las categorías del sistema/empresa dependiendo del rol del usuario	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>categories:</b> Array con las categorías resultantes.</li> </ul> </li> </ul>

Tabla A.12: API REST: GET /app/categories.

/app/categories/actives	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve todas las categorías activas del sistema/empresa dependiendo del rol del usuario	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>categories:</b> Array con las categorías resultantes.</li> </ul> </li> </ul>

Tabla A.13: API REST: GET /app/categories/actives.

/app/category/:categoryId	
Método	GET
Descripción	
El servidor devuelve la categoría cuyo id es igual a :categoryId	
Datos entrada	
Respuesta	<ul style="list-style-type: none"> <li>▪ Código: 200</li> <li>▪ Respuesta: <ul style="list-style-type: none"> <li>• category: Datos de la categoría</li> </ul> </li> </ul>

Tabla A.14: API REST: GET /app/category/:categoryId.

/app/category	
Método	POST
Descripción	
El servidor crea una nueva categoría con los datos recibidos	
Datos entrada	<ul style="list-style-type: none"> <li>▪ category: Datos de la categoría.</li> </ul>
Respuesta	<ul style="list-style-type: none"> <li>▪ Código: 200</li> <li>▪ Respuesta: <ul style="list-style-type: none"> <li>• category: Datos de la categoría guardada</li> </ul> </li> </ul>

Tabla A.15: API REST: POST /app/category.



/app/category/:categoryId	
Método	PUT
Descripción	
El servidor modifica con los datos recibidos la categoría cuyo id es igual a :categoryId	
Datos entrada	<ul style="list-style-type: none"> <li>▪ <b>category:</b> Datos a modificar de la categoría.</li> </ul>
Respuesta	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>category:</b> Categoría actualizada</li> </ul> </li> </ul>

Tabla A.16: API REST: PUT /app/category/:categoryId.

/app/category/:categoryId	
Método	DELETE
Descripción	
El servidor elimina la categoría cuyo id es igual a :categoryId	
Datos entrada	
Respuesta	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> </ul>

Tabla A.17: API REST: DELETE /app/category/:categoryId.

/app/products	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve todos los productos del sistema/empresa dependiendo del rol del usuario	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>products:</b> Array con los productos resultantes.</li> </ul> </li> </ul>

Tabla A.18: API REST: GET /app/products.

/app/products/actives	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve todos los productos activos del sistema/empresa dependiendo del rol del usuario	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>products:</b> Array con los productos resultantes.</li> </ul> </li> </ul>

Tabla A.19: API REST: GET /app/products/actives.

/app/products/category/:categoriaId	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve todos los productos filtrados por la categoría igual a :categoriaId	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>products:</b> Array con los productos resultantes.</li> </ul> </li> </ul>

Tabla A.20: API REST: GET /app/products/category/:categoriaId.

/app/product/:productId	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve el producto cuyo id es igual a :productId	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>product:</b> Datos del producto</li> </ul> </li> </ul>

Tabla A.21: API REST: GET /app/product/:productId.

/app/product	
Método	POST
Descripción	
El servidor crea un nuevo producto con los datos recibidos	
Datos entrada	<ul style="list-style-type: none"> <li>▪ <b>product:</b> Datos del producto.</li> </ul>
Respuesta	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>product:</b> Datos del producto guardado</li> </ul> </li> </ul>

Tabla A.22: API REST: POST /app/product.

/app/product/:productId	
Método	PUT
Descripción	
El servidor modifica con los datos recibidos el producto cuyo id es igual a :productId	
Datos entrada	<ul style="list-style-type: none"> <li>▪ <b>product:</b> Datos a modificar del producto.</li> </ul>
Respuesta	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>product:</b> producto actualizado</li> </ul> </li> </ul>

Tabla A.23: API REST: PUT /app/product/:productId.

/app/product/:productId	
<b>Método</b>	<b>DELETE</b>
<b>Descripción</b>	
El servidor elimina el producto cuyo id es igual a :productId	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>■ <b>Código:</b> 200</li> </ul>

Tabla A.24: API REST: DELETE /app/product/:productId.

/app/locations	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve todas las ubicaciones del sistema/empresa dependiendo del rol del usuario	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>■ <b>Código:</b> 200</li> <li>■ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>● <b>locations:</b> Array con las ubicaciones resultantes.</li> </ul> </li> </ul>

Tabla A.25: API REST: GET /app/locations.

/app/locations/actives	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve todas las ubicaciones activas del sistema/empresa dependiendo del rol del usuario	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>locations:</b> Array con las ubicaciones resultantes.</li> </ul> </li> </ul>

Tabla A.26: API REST: GET /app/locations/actives.

/app/location/:locationId	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve la ubicación cuyo id es igual a :locationId	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>location:</b> Datos de la ubicación</li> </ul> </li> </ul>

Tabla A.27: API REST: GET /app/location/:locationId.

/app/location	
Método	POST
Descripción	
El servidor crea una nueva ubicación con los datos recibidos	
Datos entrada	<ul style="list-style-type: none"> <li>▪ <b>location:</b> Datos de la ubicación.</li> </ul>
Respuesta	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>location:</b> Datos de la ubicación guardada</li> </ul> </li> </ul>

Tabla A.28: API REST: POST /app/location.

/app/location/:locationId	
Método	PUT
Descripción	
El servidor modifica con los datos recibidos la ubicación cuyo id es igual a :locationId	
Datos entrada	<ul style="list-style-type: none"> <li>▪ <b>location:</b> Datos a modificar de la ubicación.</li> </ul>
Respuesta	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>location:</b> ubicación actualizada</li> </ul> </li> </ul>

Tabla A.29: API REST: PUT /app/location/:locationId.

/app/location/:locationId	
<b>Método</b>	<b>DELETE</b>
<b>Descripción</b>	
El servidor elimina la ubicación cuyo id es igual a :locationId	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> </ul>

Tabla A.30: API REST: DELETE /app/location/:locationId.

/app/orders	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve todas las compras del sistema/empresa dependiendo del rol del usuario	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>orders:</b> Array con las compras resultantes.</li> </ul> </li> </ul>

Tabla A.31: API REST: GET /app/orders.



/app/order/:orderId	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve la compra cuyo id es igual a :orderId	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>order:</b> Datos de la compra</li> </ul> </li> </ul>

Tabla A.32: API REST: GET /app/order/:orderId.

/app/order	
<b>Método</b>	<b>POST</b>
<b>Descripción</b>	
El servidor crea una nueva compra con los datos recibidos	
<b>Datos entrada</b>	<ul style="list-style-type: none"> <li>▪ <b>order:</b> Datos de la compra.</li> </ul>
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>order:</b> Datos de la compra guardada</li> </ul> </li> </ul>

Tabla A.33: API REST: POST /app/order.

/app/order/:orderId	
Método	PUT
Descripción	
El servidor modifica con los datos recibidos la compra cuyo id es igual a :orderId	
Datos entrada	<ul style="list-style-type: none"> <li>▪ <b>order:</b> Datos a modificar de la compra.</li> </ul>
Respuesta	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>order:</b> compra actualizada</li> </ul> </li> </ul>

Tabla A.34: API REST: PUT /app/order/:orderId.

/app/order/:orderId	
Método	DELETE
Descripción	
El servidor elimina la compra cuyo id es igual a :orderId	
Datos entrada	
Respuesta	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> </ul>

Tabla A.35: API REST: DELETE /app/order/:orderId.

/app/users	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve todos los usuarios del sistema/empresa dependiendo del rol del usuario	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>users:</b> Array con los usuarios resultantes.</li> </ul> </li> </ul>

Tabla A.36: API REST: GET /app/users.

/app/user/:userId	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve el usuario cuyo id es igual a :userId	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>user:</b> Datos del usuario</li> </ul> </li> </ul>

Tabla A.37: API REST: GET /app/user/:userId.

/app/user	
<b>Método</b>	<b>POST</b>
<b>Descripción</b>	
El servidor crea un nuevo usuario con los datos recibidos	
<b>Datos entrada</b>	<ul style="list-style-type: none"> <li>▪ <b>user:</b> Datos del usuario.</li> </ul>
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>user:</b> Datos del usuario guardado</li> </ul> </li> </ul>

Tabla A.38: API REST: POST /app/user.

/app/user	
<b>Método</b>	<b>GET</b>
<b>Descripción</b>	
El servidor devuelve los datos del usuario que ha realizado la petición	
<b>Datos entrada</b>	
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>user:</b> Datos del usuario que ha realizado la petición</li> </ul> </li> </ul>

Tabla A.39: API REST: GET /app/user.

/app/user/:userId	
Método	PUT
Descripción	
El servidor modifica con los datos recibidos el usuario cuyo id es igual a :userId	
Datos entrada	<ul style="list-style-type: none"> <li>▪ <b>user:</b> Datos a modificar del usuario.</li> </ul>
Respuesta	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> <li>▪ <b>Respuesta:</b> <ul style="list-style-type: none"> <li>• <b>user:</b> usuario actualizado</li> </ul> </li> </ul>

Tabla A.40: API REST: PUT /app/user/:userId.

/app/user/:userId	
Método	DELETE
Descripción	
El servidor elimina el usuario cuyo id es igual a :userId	
Datos entrada	
Respuesta	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> </ul>

Tabla A.41: API REST: DELETE /app/user/:userId.

/app/updatePassword	
<b>Método</b>	<b>PUT</b>
<b>Descripción</b>	
El servidor modifica la contraseña del usuario que ha realizado la petición	
<b>Datos entrada</b>	<ul style="list-style-type: none"> <li>▪ <b>password:</b> Nueva contraseña.</li> </ul>
<b>Respuesta</b>	<ul style="list-style-type: none"> <li>▪ <b>Código:</b> 200</li> </ul>

Tabla A.42: API REST: PUT /app/updatePassword.

## B | Prototipo Cliente

### B.1. *Login* y pantalla principal

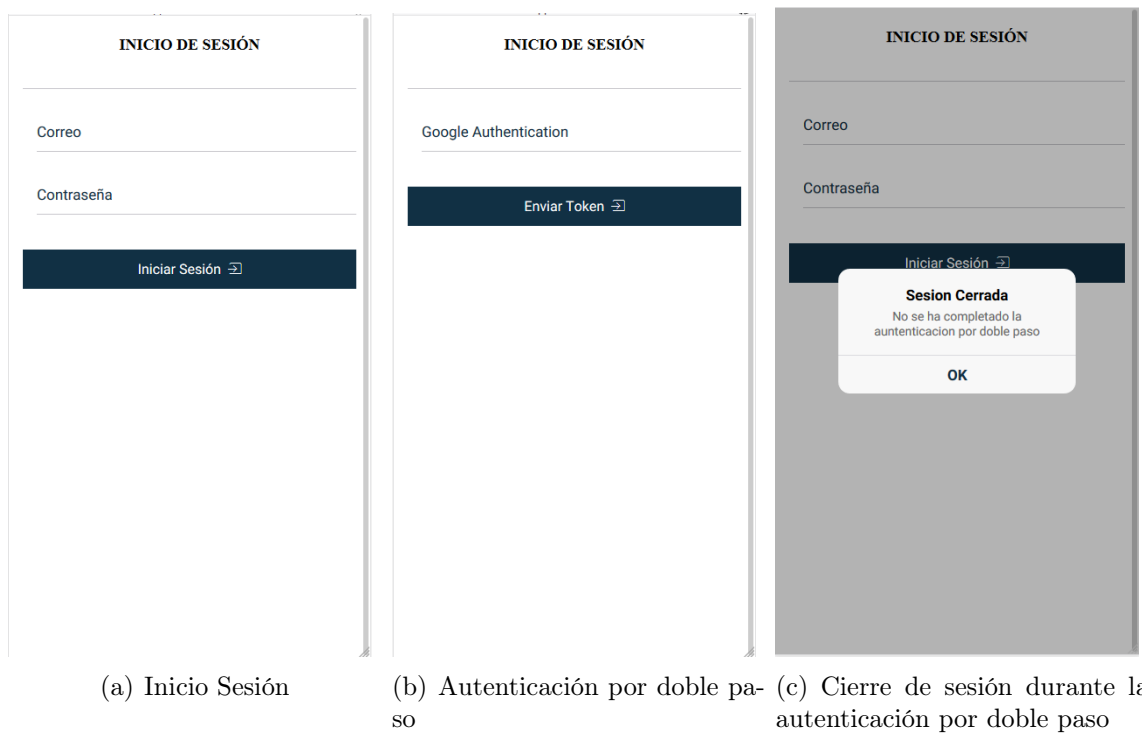


Figura B.1: Inicio de sesión.

## B.2. Inicio

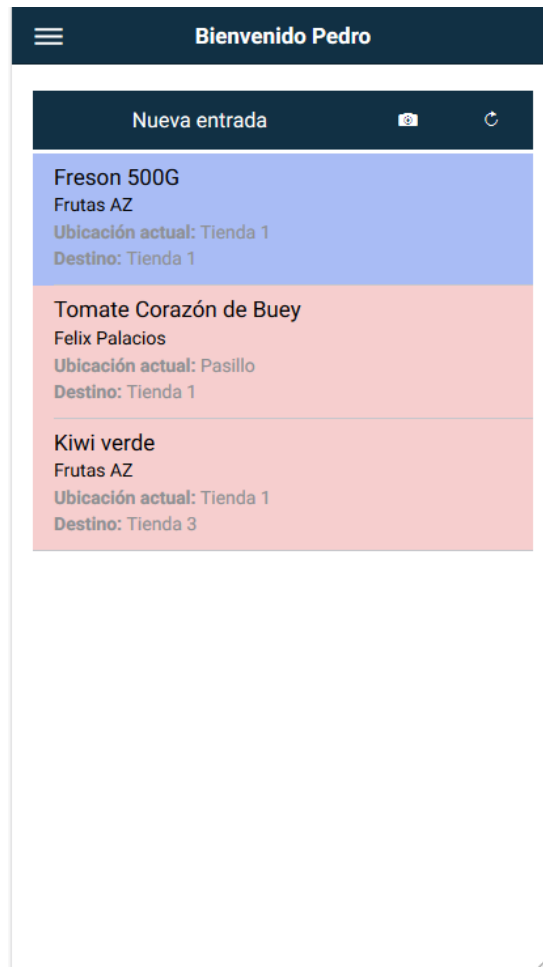


Figura B.2: Inicio.



### B.3. Crear/Modificar una entrada

(a) Selección de la categoría

(b) Selección del producto

(c) Completar datos de la entrada

(d) Entrada creada

(e) Marcar entrada en destino final

(f) Entrada en destino final

Figura B.3: Crear/Modificar una entrada.

## B.4. Usuarios

**Usuarios**

Crear Usuario

Empleado 1 User	Editar	Eliminar
Ale Ruiz Admin	Editar	Eliminar
Pedro Ruiz SuperAdmin	Editar	Eliminar
Pablo Ruiz Admin	Editar	Eliminar

**Crear Usuario**

Nombre

Empleado 2

Apellidos

Correo

empleado

DNI

Calle

Rol

User

Contraseña

••••••••

Empresa

Perove

Crear Usuario

(a) Usuarios

(b) Crear/Editar un Usuario

Figura B.4: Usuarios.

## B.5. Empresas

The figure displays two mobile application screens for managing companies.

**(a) Empresas**: This screen features a dark blue header with a back arrow and the title "Empresas". Below the header is a dark blue button labeled "Crear Empresa". The main content area shows a list of companies. The first entry is "Perove" with the CIF "916221494" and the name "Jorge Guillen 4". To the right of the CIF are two buttons: "Editar" and "Eliminar".

**(b) Crear/Editar una empresa**: This screen has a dark blue header with a back arrow and the title "Crear Empresa". The form contains five input fields: "Nombre", "Telefono" (with a dropdown arrow), "CIF", "Dirección", and "Description". At the bottom is a dark blue button labeled "Crear Empresa".

(a) Empresas

(b) Crear/Editar una empresa

Figura B.5: Empresas.

## B.6. Proveedores

Proveedores

Crear Proveedor

Frutas AZ	Editar	Eliminar
Frutas Tono	Editar	Eliminar
Frutas El Amigo	Editar	Eliminar

Back Crear Proveedor

Nombre

Felix Palacios

Telefono

915077225

CIF

A78583499

Dirección

Ctra. Villaverde A Vallecas Km. 3,8

Description

Crear Proveedor

(a) Proveedores

(b) Crear/Editar un proveedor

Figura B.6: Proveedores.

## B.7. Categorías



Figura B.7: Categorías.

## B.8. Productos

Productos

Crear Producto

Freson 500G	<input checked="" type="checkbox"/>	Editar	Eliminar
Freson	<input checked="" type="checkbox"/>	Editar	Eliminar
Aliño de mi pue...	<input checked="" type="checkbox"/>	Editar	Eliminar
Caldo cocido	<input checked="" type="checkbox"/>	Editar	Eliminar
Caldo pollo	<input checked="" type="checkbox"/>	Editar	Eliminar
Frambuesa	<input checked="" type="checkbox"/>	Editar	Eliminar
Kiwi verde	<input checked="" type="checkbox"/>	Editar	Eliminar
cocacola	<input checked="" type="checkbox"/>	Editar	Eliminar
cerveza	<input checked="" type="checkbox"/>	Editar	Eliminar
Acelga	<input checked="" type="checkbox"/>	Editar	Eliminar
Patata Elody	<input checked="" type="checkbox"/>	Editar	Eliminar
Patata Monalisa	<input checked="" type="checkbox"/>	Editar	Eliminar
Cebolla	<input checked="" type="checkbox"/>	Editar	Eliminar
Cebolla Dulce	<input checked="" type="checkbox"/>	Editar	Eliminar

< Back

Editar Producto

Nombre

Freson 500G

Codigo

Categoria

Berris

Activo

☒

Editar Producto

(a) Productos

(b) Crear/Editar un producto

Figura B.8: Productos.

## B.9. Ubicaciones

Ubicaciones

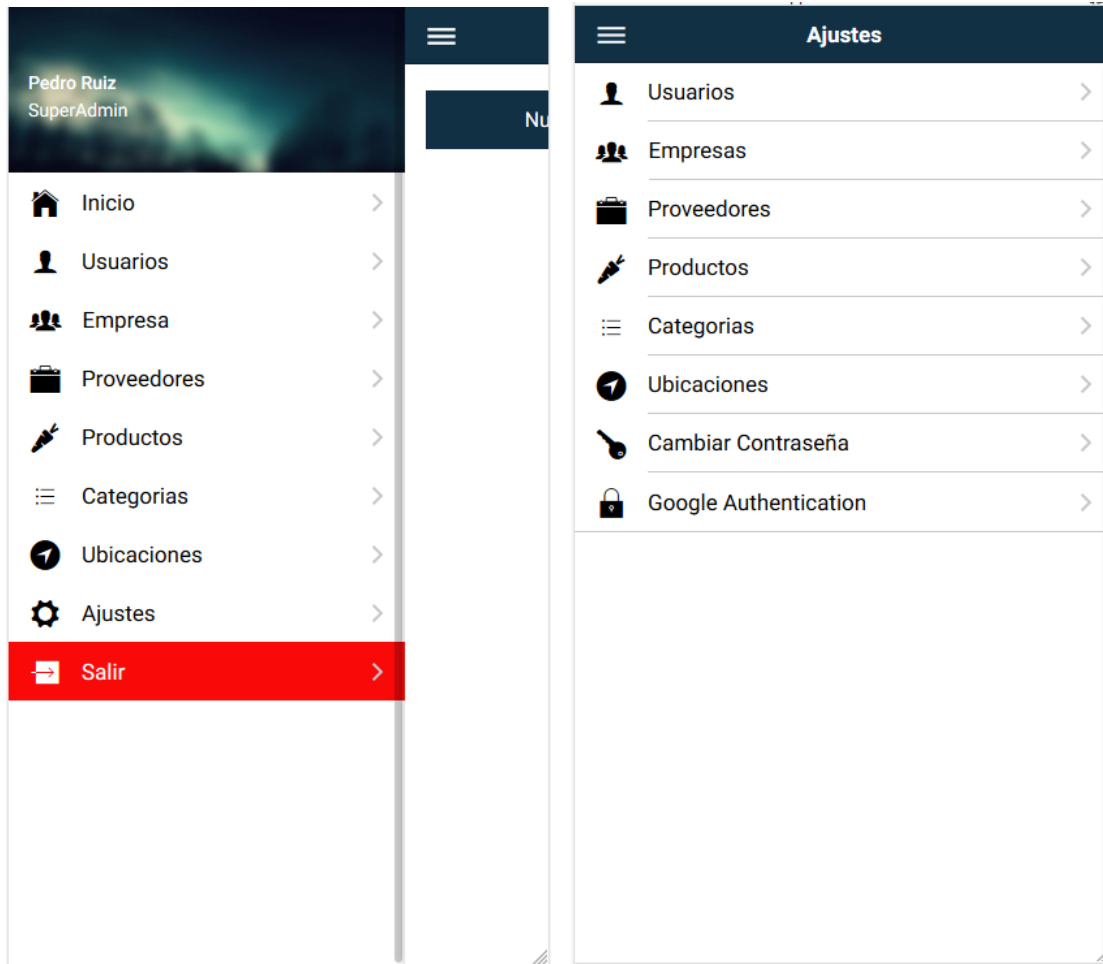
Nueva ubicación

Crear Ubicación

Tienda 1	<input checked="" type="checkbox"/>	Eliminar
Tienda 2	<input checked="" type="checkbox"/>	Eliminar
Tienda 3	<input checked="" type="checkbox"/>	Eliminar
Nota	<input checked="" type="checkbox"/>	Eliminar
Comprado	<input checked="" type="checkbox"/>	Eliminar
Pasillo	<input checked="" type="checkbox"/>	Eliminar
Proveedor	<input checked="" type="checkbox"/>	Eliminar

Figura B.9: Ubicaciones.

## B.10. Ajustes



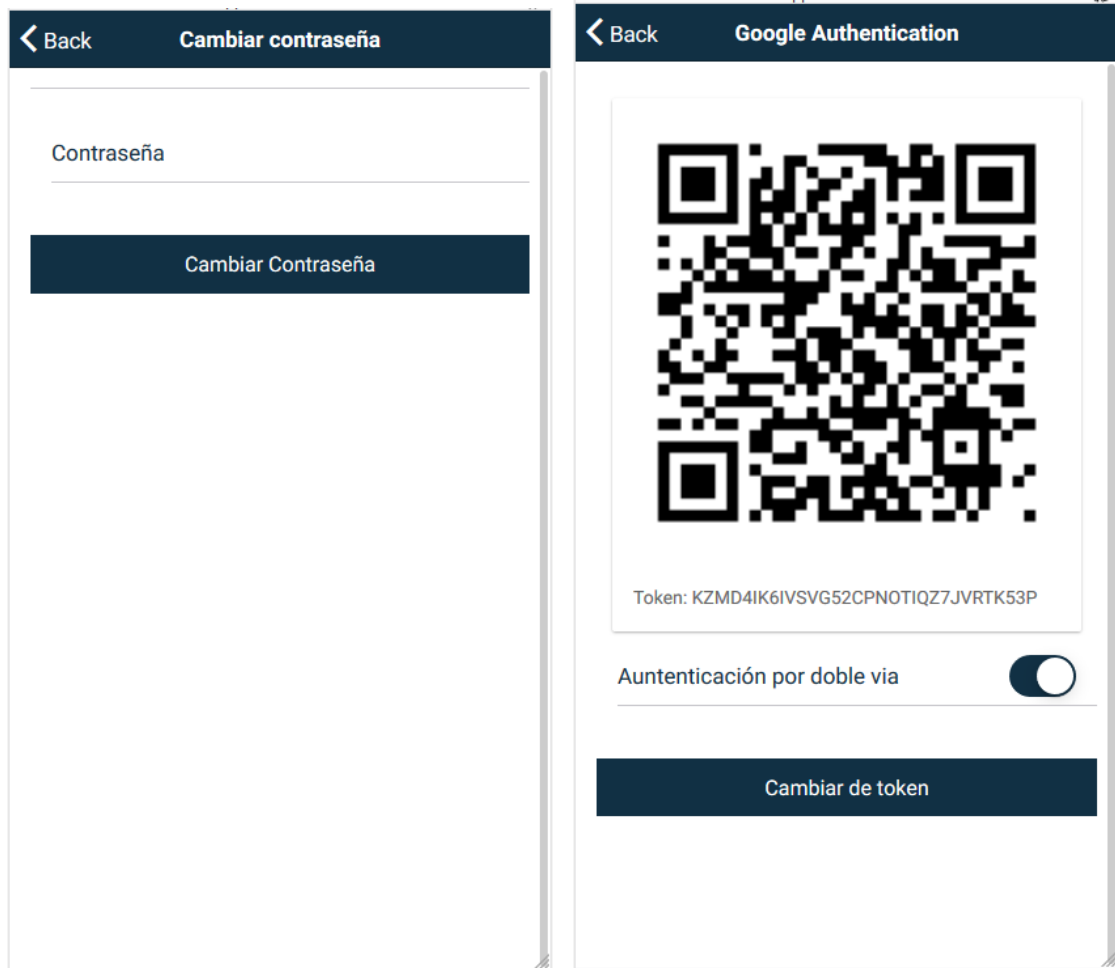
(a) Barra Ajustes

(b) Menú Ajustes

Figura B.10: Ajustes.



## B.11. Autenticación



(a) Cambiar contraseña

(b) Cambiar token doble paso

Figura B.11: Cambiar contraseña/token doble paso.



# Bibliografía

- [1] S. Huseby, *Innocent Code: A Security Wake-Up Call for Web Programmers*. Wiley, 2004, ISBN: 9780470857472. dirección: <https://books.google.es/books?id=RjVjgPQsKogC> (vid. págs. 1, 29).
- [2] *Número global de ataques web bloqueados por día desde 2015 hasta 2017 (en miles)*, Último Acceso 10 de junio de 2018. dirección: <https://www.statista.com/statistics/494961/web-attacks-blocked-per-day-worldwide/> (vid. pág. 1).
- [3] K. Duuna, *Secure Your Node.js Web Application: Keep Attackers Out and Users Happy*, ép. The pragmatic programmers. Pragmatic Bookshelf, 2016, ISBN: 9781680500851. dirección: <https://books.google.es/books?id=AofssgEACAAJ> (vid. págs. 2, 3).
- [4] B. Sullivan, «Server-side JavaScript injection», *Black Hat USA*, 2011 (vid. pág. 2).
- [5] *Index TIOBE, Top lenguajes de Programación*, Último Acceso 1 de enero de 2018. dirección: <https://www.tiobe.com/tiobe-index/> (vid. pág. 3).
- [6] J. Wilken y A. Bradley, *Ionic in Action: Hybrid Mobile Apps with Ionic and AngularJS*. Manning Publications, 2015, ISBN: 9781633430082. dirección: <https://books.google.es/books?id=jyUgrgEACAAJ> (vid. pág. 3).
- [7] *Open Web Application Security Project*. dirección: <https://www.owasp.org> (vid. págs. 3, 47).
- [8] J. Vacca, *Los secretos de la seguridad en Internet*, ép. Top Secret. Anaya Multimedia, 1997, ISBN: 9788441500952. dirección: <https://books.google.es/books?id=poH5PAAACAAJ> (vid. pág. 5).
- [9] B. Laurie, A. Langley y E. Kasper, *Certificate Transparency*, RFC 6962, jun. de 2013. DOI: 10.17487/RFC6962. dirección: <https://rfc-editor.org/rfc/rfc6962.txt> (vid. págs. 5, 33).
- [10] *Repositorios activos en github*, Último Acceso 1 de enero de 2018. dirección: <http://githut.info/> (vid. pág. 6).
- [11] *Entendiendo Event Loop*, Último Acceso 10 de junio de 2018. dirección: <https://aprendiendo-nodejs.blogspot.com/2017/07/entendiendo-el-event-loop.html> (vid. pág. 8).
- [12] *Web Content Accessibility Guidelines (WCAG) 2.0*, 2018. dirección: <https://www.w3.org/TR/WCAG20/> (vid. pág. 21).
- [13] A. Shostack, *Threat modeling: Designing for security*. John Wiley & Sons, 2014 (vid. pág. 29).
- [14] OWASP, *Testing for NoSQL injection*, Último Acceso 10 de junio de 2018. dirección: [https://www.owasp.org/index.php/Testing\\_for\\_NoSQL\\_injection](https://www.owasp.org/index.php/Testing_for_NoSQL_injection) (vid. pág. 29).

- [15] J. R. Vacca y J. R. Vacca, *Computer and Information Security Handbook, Second Edition*, 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013, ISBN: 0123943973, 9780123943972 (vid. pág. 32).
- [16] *Estandar JSON Web Token (JWT), rfc7519*, dirección: <https://tools.ietf.org/html/rfc7519> (vid. pág. 35).
- [17] *NIST is No Longer Recommending Two-Factor Authentication Using SMS*, Último Acceso 10 de junio de 2018. dirección: [https://www.schneier.com/blog/archives/2016/08/nist\\_is\\_no\\_long.html](https://www.schneier.com/blog/archives/2016/08/nist_is_no_long.html) (vid. pág. 38).
- [18] N. Provos y D. Mazieres, «A Future-Adaptable Password Scheme.», en *USENIX Annual Technical Conference, FREENIX Track*, 1999, págs. 81-91 (vid. pág. 39).
- [19] OWASP, *OWASP Top 10 Application Security Risks - 2017*, Último Acceso 10 de junio de 2018. dirección: [https://www.owasp.org/index.php/Top\\_10-2017\\_Top\\_10](https://www.owasp.org/index.php/Top_10-2017_Top_10) (vid. pág. 41).
- [20] A. Cavoukian y M. Dixon, *Privacy and Security by Design: An Enterprise Architecture Approach*. Ontario: Information y Privacy Commissioner, 2013 (vid. pág. 48).
- [21] J. Diaz, D. Arroyo y F. B. Rodriguez, «On securing online registration protocols: Formal verification of a new proposal», *Knowledge-Based Systems*, vol. 59, págs. 149-158, 2014 (vid. pág. 48).
- [22] J. M. Zuriarrain, *Dropbox reconoce el 'hacking' de 60 millones de cuentas: cómo saber si la tuya está afectada*, 2016. dirección: [https://elpais.com/tecnologia/2016/08/31/actualidad/1472642567\\_500051.html](https://elpais.com/tecnologia/2016/08/31/actualidad/1472642567_500051.html).
- [23] G. Saini, *Hybrid Mobile Development with Ionic*. Packt Publishing, 2017, ISBN: 1785286056, 9781785286056.
- [24] A. Sanchez-Gomez, J. Diaz, L. Hernandez-Encinas y D. Arroyo, *Review of the Main Security Threats and Challenges in Free-Access Public Cloud Storage Servers*. Springer, 2018, págs. 263-281.
- [25] J. Reynolds, T. Smith, K. Reese, L. Dickinson, S. Ruoti y K. Seamons, «A Tale of Two Studies: The Best and Worst of YubiKey Usability», en *2018 IEEE Symposium on Security and Privacy (SP)*, vol. 00, págs. 1090-1106. DOI: 10.1109/SP.2018.00067. dirección: [doi.ieeecomputersociety.org/10.1109/SP.2018.00067](https://doi.ieeecomputersociety.org/10.1109/SP.2018.00067).